INTERNATIONAL STANDARD

ISO/IEC 14515-1

First edition
2000-12-15

AMENDMENT 1
2003-12-15

IEEE Std 2003.1b-2000
(Amendment to
IEEE Std 2003.1-1992)

Information technology — Portable Operating System Interface (POSIX®) — Test methods for measuring conformance to POSIX —

Part 1: System interfaces

AMENDMENT 1: Realtime Extension (C Language)

Technologies de l'information — Interface de système de fonctionnement portable (POSIX®) — Méthodes d'essai pour mesurer la conformité au POSIX —

Partie 1: Interfaces de système

AMENDEMENT 1: Extension en temps réel (langage C)

Reference number
ISO/IEC 14515-1:2000/Amd.1:2003(E)
IEEE Std 2003.1b-2000
(Amendment to IEEE Std 2003.1-1992)

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

ISO/IEC 14515-1:2000/Amd.1:2003(E)
IEEE Std 2003.1b-2000
(Amendment to IEEE Std 2003.1-1992)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ISO/IEC 14515-1:2000/Amd.1:2003(E)
IEEE Std 2003.1b™-2000
(Amendment to IEEE Std 2003.1™-1992)

Information Technology—Portable Operating System Interface (POSIX®)—Part 1: Test method for measuring conformance to POSIX

Amendment 1: Realtime Extension (C Language)

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

Approved 30 March 2000
IEEE-SA Standards Board

Approved 2 November 2000
American National Standards Institute

Adopted as an International Standard by the International Organization for Standardization and by the International Electrotechnical Commission

Published by The Institute of Electrical and Electronics Engineers, Inc.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Abstract: This standard defines the test method specifications for IEEE Std 1003.b-1993 (based on the document corresponding to the merger of IEEE Std 1003.1-1990 and IEEE Std 1003.1b-1993). The test method specifications consist of assertions to be tested and related test procedures. As an amendment to IEEE Std 1003.1-1990, this standard is structured to amend those portions of IEEE Std 2003.1-1992 (the test method specification for IEEE Std 1003.1-1990) that correspond to the amended parts of IEEE Std 1003.1-1990. This standard is aimed primarily at providers of test methods for IEEE Std 1003.1b-1993 and at implementors of IEEE Std 1003.1b-1993.

Keywords: assertion, assertion test, C programming language, POSIX, POSIX Conformance Document, POSIX Conformance Test Procedure, POSIX Conformance Test Suite, realtime, test method specification, test result code

---

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2003 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 15 August 2003. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Inc.

POSIX is a registered trademark of the Institute of Electrical and Electronics Engineers, Inc.

Print: ISBN 0-7381-3759-6 SH95152
PDF: ISBN 0-7381-3760-X SS95152

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

--`,`,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

International Standard ISO/IEC 14515-1:2000/Amd.1:2003(E)

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 1 to ISO/IEC 14515-1:2000 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.



International Organization for Standardization / International Electrotechnical Commission
Case postale 56 • CH-1211 Genève 20 • Switzerland

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS.**"

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Contents

Introduction viii
Organization of This Standard viii
How to Read This Standard viii
Related Standards Activities ix

Section 1: General 1
1.1 Scope 1
1.2 Normative References 2
1.3 Conformance 3
1.4 Test Methods 4

Section 2: Terminology and General Requirements 19
2.1 Conventions 19
2.2 Definitions 20
2.3 General Concepts 28
2.4 Error Numbers 34
2.5 Primitive System Data Types 35
2.6 Environment Description 35
2.7 C Language Definitions 37
2.8 Numerical Limits 40
2.9 Symbolic Constants 43

Section 3: Process Primitives 48
3.1 Process Creation and Execution 48
3.1.1 Process Creation 48
3.1.2 Execute a File 50
3.2 Process Termination 53
3.2.1 Wait for Process Termination 53
3.2.2 Terminate a Process 53
3.3 Signals 55
3.3.1 Signal Concepts 55
3.3.2 Send a Signal to a Process 65
3.3.3 Manipulate Signal Sets 65
3.3.4 Examine and Change Signal Action 65
3.3.5 Examine and Change Blocked Signals 67
3.3.6 Examine Pending Signals 67
3.3.7 Wait for a Signal 68
3.3.8 Synchronously Accept a Signal 68
3.3.9 Queue a Signal to a Process 72
3.4 Timer Operations 75
3.4.1 Schedule Alarm 76
3.4.2 Suspend Process Execution 76
3.4.3 Delay Process Execution 76

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 4: Process Environment 78
4.8 Configurable System Variables 78
4.8.1 Get Configurable System Variables 78

Section 5: Files and Directories 82
5.1 Directories 82
5.1.1 Format of Directory Entries 82
5.1.2 Directory Operations 82
5.2 Working Directory 82
5.2.1 Change Current Working Directory 82
5.2.2 Get Working Directory Pathname 83
5.3 General File Creation 83
5.3.1 Open a File 83
5.3.2 Create New File or Rewrite an Existing One 84
5.3.3 Set File Creation Mask 84
5.3.4 Link to a File 85
5.4 Special File Creation 85
5.4.1 Make a Directory 85
5.4.2 Make a FIFO Special File 85
5.5 File Removal 86
5.5.1 Remove Directory Entries 86
5.5.2 Remove a Directory 86
5.5.3 Rename a File 86
5.6 File Characteristics 87
5.6.1 File Characteristics: Header and Data Structure 87
5.6.2 Get File Status 88
5.6.3 Check File Accessibility 89
5.6.4 Change File Modes 89
5.6.5 Change Owner and Group of a File 92
5.6.6 Set File Access and Modification Times 93
5.6.7 Truncate a File to a Specified Length 93
5.7 Configurable Pathname Variables 96
5.7.1 Get Configurable Pathname Variables 96

Section 6: Input and Output Primitives 99
6.1 Pipes 99
6.1.1 Create an Inter-Process Channel 99
6.2 File Descriptor Manipulation 99
6.2.1 Duplicate an Open File Descriptor 99
6.3 File Descriptor Deassignment 100
6.3.1 Close a File 100
6.4 Input and Output 101
6.4.1 Read from a File 101
6.4.2 Write to a File 102
6.5 Control Operations on Files 103
6.5.1 Data Definitions for File Control Operations 103
6.5.2 File Control 103
6.5.3 Reposition Read/Write File Offset 106
6.6 File Synchronization 107

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.6.1 Synchronize the State of a File 107
6.6.2 Synchronize the Data of a File 110
6.7 Asynchronous Input and Output 112
6.7.1 Data Definitions for Asynchronous Input and Output 112
6.7.2 Asynchronous Read 115
6.7.3 Asynchronous Write 121
6.7.4 List Directed I/O 128
6.7.5 Retrieve Error Status of Asynchronous I/O Operation 138
6.7.6 Retrieve Return Status of Asynchronous I/O Operation 140
6.7.7 Cancel Asynchronous I/O Request 141
6.7.8 Wait for Asynchronous I/O Request 144
6.7.9 Asynchronous File Synchronization 147

Section 7: Device- and Class- Specific Functions 153

Section 8: Language-Specific Services for the C Programming Language 154
8.2.2 Open a Stream on a File Descriptor 154

Section 9: System Databases 155

Section 10: Data Interchange Format 156

Section 11: Synchronization 157
11.1 Semaphore Characteristics 157
11.2 Semaphore Functions 158
11.2.1 Initialize an Unnamed Semaphore 158
11.2.2 Destroy an Unnamed Semaphore 161
11.2.3 Initialize/Open a Named Semaphore 162
11.2.4 Close a Named Semaphore 168
11.2.5 Remove a Named Semaphore 170
11.2.6 Lock a Semaphore 172
11.2.7 Unlock a Semaphore 180
11.2.8 Get the Value of a Semaphore 184

Section 12: Memory Management 187
12.1 Memory Locking Functions 189
12.1.1 Lock/Unlock the Address Space of a Process 189
12.1.2 Lock/Unlock a Range of Process Address Space 193
12.2 Memory Mapping Functions 197
12.2.1 Map Process Address to a Memory Object 197
12.2.3 Change Memory Protection 207
12.2.4 Memory Object Synchronization 211
12.3 Shared Memory Functions 215
12.3.1 Open a Shared Memory Object 215
12.3.2 Remove a Shared Memory Object 222

Section 13: Execution Scheduling 225
13.1 Scheduling Parameters 225

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.2 Scheduling Policies 225
13.2.1 SCHED_FIFO 226
13.2.2 SCHED_RR 228
13.2.3 SCHED_OTHER 228
13.3 Process Scheduling Functions 229
13.3.1 Set Scheduling Parameters 229
13.3.2 Get Scheduling Parameters 232
13.3.3 Set Scheduling Policy and Scheduling Parameters 234
13.3.4 Get Scheduling Policy 238
13.3.5 Yield Processor 239
13.3.6 Get Scheduling Parameter Limits 240

Section 14: Clocks and Timers 245
14.1 Data Definitions for Clocks and Timers 245
14.1.1 Time Value Specification Structures 245
14.1.2 Timer Event Notification Control Block 246
14.1.3 Type Definitions 246
14.1.4 Manifest Constants 246
14.2 Clocks and Timer Functions 247
14.2.1 Clocks 247
14.2.2 Create a Per-Process Timer 253
14.2.3 Delete a Per-Process Timer 257
14.2.4 Per-Process Timers 259
14.2.5 High Resolution Sleep 265

Section 15: Message Passing 269
15.1 Data Definitions for Message Queues 269
15.1.1 Data Structures 269
15.2 Message Passing Functions 270
15.2.1 Open a Message Queue 270
15.2.2 Close a Message Queue 278
15.2.3 Remove a Message Queue 280
15.2.4 Send a Message to a Message Queue 282
15.2.5 Receive a Message From a Message Queue 285
15.2.6 Notify Process that a Message is Available on a Queue 287
15.2.7 Set Message Queue Attributes 289
15.2.8 Get Message Queue Attributes 291

Annex A
(normative)

Conforming Test Results 293
A.1 General 293
A.2 Terminology and General Requirements 293
A.3 Process Primitives 295
A.4 Process Environment 302
A.5 Files and Directories 302
A.6 Input and Output Primitives 306
A.7 Device- and Class-Specific Functions 314
A.8 Language-Specific Services for the C Programming Language 314

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.9 System Databases 314
A.10 Data Interchange Format 314
A.11 Synchronization 314
A.12 Memory Management 319
A.13 Execution Scheduling 326
A.14 Clocks and Timers 330
A.15 Message Passing 335

Annex B
(informative)

Bibliography 341
B.1 Related Open Systems Standards 341
B.2 Other Standards 343
B.3 Historical Documentation and Introductory Texts 343
B.4 Other Sources of Information 345

Identifier Index 347

Alphabetic Topical Index 350

TABLES

Table 1-1 – PCTS Symbols and Values 6
Table 2-1 – Minimum Values 40
Table 2-2 – Run-Time Invariant Values (Possibly Indeterminate) 42
Table 2-3 – Maximum Values 43
Table 2-4 – Compile-Time Symbolic Constants 44
Table 2-5 – Execution-Time Symbolic Constants 46
Table 3-1 – Required Signals 55
Table 3-2 – Job Control Signals 56
Table 3-3 – Memory Protection Signals 56
Table 4-2 – Configurable System Variables 80

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

(This Introduction is not a normative part of IEEE Std 2003.1b-2000, IEEE Standard for Information Technology–Test Methods Specifications for Measuring Conformance to POSIX-Part 1: System Application Program Interface(API)-Amendment 1: Realtime Extension [C Language]).

This standard defines the test method specifications for IEEE Std 1003.1b-1993 (based on the document corresponding to the merger of IEEE Std 1003.1-1990 and IEEE Std 1003.1b-1993). The test method specifications consist of assertions to be tested and related test procedures. As an amendment to IEEE Std 1003.1-1990, this standard is structured to amend those portions of IEEE Std. 2003.1-1992 {4} (the test method specification for IEEE Std 1003.1-1990) that correspond to the amended parts of IEEE Std 1003.1-1990. This standard is aimed primarily at providers of test methods for IEEE Std 1003.1b-1993 and at implementors of IEEE Std 1003.1b-1993.

Organization of This Standard

This document is organized into five parts, as follows:

(1) Statement of scope, normative references, conformance requirements, and test methods (Section 1)

(2) Conventions and definitions (Section 2)

(3) Assertions to test POSIX.1b {3} (Sections 2 through 15)

(4) Conforming test results (Annex A)

(5) Bibliography (Annex B)

This introduction, any footnotes, notes accompanying the test, and the *informative* annexes are not considered part of this standard. Annex A is normative. Annex B is informative.

How to Read This Standard

This document is organized using the same section numbering as POSIX.1b {3}, in order to facilitate finding the testing requirements for a particular element of POSIX.1b {3}. Subclause 1.4 has been added to this document to keep the section numbering consistent with POSIX.1b {3} and to provide a place to describe features relevant to the test methods. It is strongly recommended that the reader review 1.4 after reading 1.1, 1.2, and this introduction. Where possible, this document tries to use the same assertion numbering that was used as IEEE Std 2003.1-1992 {4}. This document skips over the assertion numbers where there are no assertions that correspond to an assertion in IEEE Std 2003.1-1992 {4}. The assertions have been numbered with a fractional part where more than one assertion takes the place of a single assertion in IEEE Std 2003.1-1992 {4}.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Related Standards Activities

Activities to extend this standard to address additional requirements are in progress, and similar efforts can be anticipated in the future.

The following areas are under active consideration at this time, or are expected to become active in the near future: [1)]

(1) Language-independent service descriptions of this standard

(2) C, Ada, and FORTRAN language bindings to (1)

(3) Shell and utility facilities

(4) Verification testing methods

(5) Multithreaded process facilities

(6) Secure/trusted system considerations

(7) Network interface facilities

(8) System administration

(9) Graphical user interfaces

(10) Profiles describing application or user-specific combinations of Open Systems standards for: supercomputing, multiprocessor, and batch extensions; transaction processing; realtime systems; and multiuser systems based on historical models

(11) An overall guide to POSIX-based or related Open Systems standards and profiles.

Extensions are approved as "amendments" or "revisions" to this document, following IEEEStandards procedures.

Approved amendments are published separately until the full document is reprinted; such amendments are then incorporated in their proper positions.

If you have an interest in participating in the PASC Working Groups addressing these issues, please send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, and ask to have this forwarded to the chairperson of the appropriate PASC Working Group. If you have an interest in participating in this work at the international level, contact your ISO/IEC national body.

---

1)

A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903, Telephone: +1 202 371-0101; FAX: +1 202 728-9614.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

IEEE Std 2003.1b-2000 was prepared by a breakout group in the 2003 Working Group focused on developing the IEEE 2003.1b standard, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society. At the time this standard was approved, the membership of the IEEE 2003.1b breakout groups was as follows:

Portable Applications Standards Committee

Chair:	Lowell Johnson
Vice Chair:	Joe Gwinn
Secretary:	Nick Stoughton
Functional Chairs:	Andrew Josey
	Jay Ashford
	Curtis Royster
	Jason Zions

2003 Working Group Officials

Chair:	Barry Hedquist
	Roger Martin (retired)
Vice Chair:	John Davies (2003)
	Ken Thomas (2003.1b)
Editors:	Bruce Weiner (primary)
	Jeffrey S. Haemer (supporting)
	Barry Hedquist
Secretary:	Keith Stobie (1994)

Technical Reviewers

Ted Baker	John Davies	Barry Hedquist
Jeffrey Haemer	Ken Thomas	Bruce Weiner

2003.1b Working Group

John Davies	Leo Hansen	Ken Thomas
Jeffrey Haemer	Curtis Royster	Bruce Weiner

The following members of the balloting committee voted on this standard:

Khaled Al-Ali	Barry Hedquist	William R. Smith
Andy Bihain	Lowell Johnson	Kenneth G. Thomas
Susan Corwin	Roger Martin	Bruce Weiner
Steven J. Dovitch	Gerald Powell	Andrew E. Wheeler
Michel P. Gien	Curtis Royster	Alex White
Patrick Hebert		John Zolnowsky

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When the IEEE-SA Standards Board approved this standard on 30 March 2000, it had the following membership:

Donald N. Heirman, *Chair*
James T. Carlo, *Vice Chair*
Judith Gorman, *Secretary*

Satish K. Aggarwal
Mark D. Bowman
Gary R. Engmann
Harold E. Epstein
H. Landis Floyd
Jay Forster*
Howard M. Frazier
Ruben D. Garzon
James H. Gurney
Richard J. Holleman
Lowell G. Johnson
Robert J. Kennelly
Joseph L. Koepfinger*
Peter H. Lips
L. Bruce McClung
Daleep C. Mohla
James W. Moore
Robert F. Munzner
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Akio Tojo
Donald W. Zipse

*Member Emeritus

Also included is the following nonvoting IEEE-SA Standards Board liaison:

Alan Cookson, *NIST Representative*
Donald R. Volzka, *TAB Representative*

Yvette Ho Sang
IEEE Standards Project Editor

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Information Technology — Portable Operating System Interface (POSIX®) — Test methods for measuring conformance to POSIX —

Part 1: System interfaces

Amendment 1: Realtime Extension (C Language)

Section 1: General

1.1 Scope

This standard defines the test method specifications for measuring conformance of an implementation to POSIX.1b {3}[2]. Assertions are the primary test method specification for measuring conformance to POSIX.1b {3}. Conformance to POSIX.1b {3} requires conformance to IEEE Std 1003.1b-1993 as it has been modified by POSIX.1b {3}. Therefore, the test method specifications for POSIX.1b {3} assume that the test method specifications of IEEE Std 2003.1-1992 {4} apply, except as modified by this standard.

This standard is intended for use by

(1) Developers of POSIX.1b {3} test methods

(2) Implementors of POSIX.1b {3} implementations

(3) Application writers for POSIX.1b {3} conforming implementations

(4) POSIX.1b {3} testing laboratories

(5) Others interested in validating the conformance of a vendor-claimed POSIX.1b {3} implementation

[2] The numbers in braces correspond to those of the references in 1.2.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

The purpose of this standard is to specify the assertions and related test method specifications for measuring conformance of an implementation to POSIX.1b {3}.

This standard specifies additions and modifications to IEEE Std 2003.1-1992 {4}.

Testing conformance of an implementation to a standard includes testing the claimed capabilities and behavior of the implementation with respect to the conformance requirements of the standard. These test methods are intended to provide a reasonable, practical assurance that the implementation conforms to the standard. Use of these test methods will not guarantee conformance of an implementation to POSIX.1b {3}; that normally would require exhaustive testing, which is impractical for both technical and economic reasons.

IEEE Std 2003.1b-2000 defines a means of measuring conformance to the POSIX.1b {3} technical specifications. Any question related to interpretation of technical specifications arising from the use of this standard is a question of interpretation of POSIX.1b {3}.

1.2 Normative References

The following standards contain provisions which, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

{1} ISO/IEC 646: 1991, *Information processing—ISO 7-bit coded character set for information interchange* [3].

{2} ISO/IEC 9899: 1990, *Programming languages—C.*

{3} IEEE Std 1003.1b-1993, *IEEE Standard for Information Technology—POSIX—Part 1: System Application Program Interface (API)—Amendment 1: Realtime Extension [C Language].*

{4} IEEE Std 2003.1-1992, *IEEE Standard for Information Technology—Test Methods for Measuring Conformance to POSIX—Part 1: System Interfaces.*

{5} ISO/IEC 13210:1999 (IEEE Std 2003-1997), *Information technology—Requirements and Guidelines for Test Method Specifications and Test Method Implementations for Measuring Conformance to POSIX Standards.*

---

[3] ISO/IEC documents are available from the ISO Central Secretariat, Case Postal 56, 1 rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/). ISO/IEC publications are available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112, USA (http://global.ihs.com/). Electronic copies are available in the United States from the American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA (http://www.ansi.org/).

Copyright © 2003 IEEE – All rights reserved
--`,,`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; there are no POSIX.1b {3} assertions.

1.3.1.2 Documentation

There are only IEEE Std 2003.1 {4} assertions in this subclause; there are no POSIX. 1b {3} assertions.

1.3.1.3 Conforming Implementation Options

There are no requirements for conforming implementations in this subclause.

1.3.2 Application Conformance

There are no requirements for conforming implementations in this subclause.

1.3.3 Language -Dependent Services for the C Programming Language

There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; there are no POSIX.1b {3} assertions.

1.3.3.1 Types of Conformance

There are no requirements for conforming implementations in this subclause.

1.3.3.2 C Standard Language-Dependent System Support

There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; there are no POSIX.1b {3} assertions.

1.3.3.3 Common-Usage C Language-Dependent System Support

(IEEE Std 2003.1-1992 {4} DGA01
UNUSED

M_GD_CommonC_diffs (function) =
FOR: *function ()*
IF the implementation does not provide C Standard {2} support **THEN**
TEST: Subclause 8.1 of the POSIX.1b contains the details of all differences between the interface *function ()* provided and that required by the C Standard {2}.
ELSE *NO_OPTION*

GD_CommonC_diffs
FOR: *function* ()
M_GD_CommonC_diffs (function)
Conformance for conformance: PASS, NO_OPTION

1.3.4 Other C Language-Related Specifications

(IEEE Std 2003.1-1992 {4} GA01
UNUSED

M_GA_macro_args (function; header1; header2; header3; header4) =
IF the interface *function* () is defined as a macro **THEN**
SETUP: The headers `<header1>`, `<header2>`, `<header3>`, and `<header4>` are included.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: When the macro *function* () is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), the macro evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary.

ELSE *NO_OPTION*

GA_macro_args

FOR: All interfaces except *abort* (), *assert* (), *getc* (), *putc* (), *setjmp* (), *sig-setjmp* (), and *tzset* ().

M_GA_macro_args (function; header1; header 2; header 3; header 4)

Conformance for conformance: PASS, NO_OPTION

M_GA_macro_result_decl (function_type; function; header1; header2; header3; header4) =

IF the interface *function* () is defined as a macro **THEN**

SETUP: The headers `<header1>`, `<header2>`, `<header3>`, and `<header4>` are included.

TEST: When the macro *function* () is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), it expands to an expression with the result type *function_type*.

ELSE *NO_OPTION*

GA_macro_result_decl

FOR: All functions implemented as macros

M_GA_macro_result_decl (function; header1; header2; header3; header4)

Conformance for conformance: PASS, NO_ OPTION

1.3.5 Other Language-Related Specifications

There are no requirements for conforming implementations in this clause.

1.4 Test Methods

This clause defines conventions, terminology, testing methodology, and testing control variables used in this standard.

1.4.1 Conventions

The conventions used in specifying the assertions in this standard follow those of ISO/IEC 13210:1999 {5} with the extensions described below.

One convention used in this document is the use of *PCTS_* variables. Because of the IF... Otherwise structure in POSIX.1b {3}, it was necessary to combine the use of {_POSIX_} compile-time symbolic constants that indicate optional facilities in POSIX.1b {3} with the case where the implementor chooses to provide some of the functions specified to be present when the option is defined. For example, an implementation that provides the *sem_init*() function, but does not provide all facilities required for the {_POSIX_SEMAPHORES_} option to be set, would require a PCTS to be configured with the constant *PCTS_sem_init* to be **TRUE**. (Note, that this constant would also need to be **TRUE** if the {_POSIX_SEMAPHORES_} option is set.)

The following construct means that assertion 03, within the corresponding clause of IEEE Std 2003.1-1992 {4}, is not used by this standard:

(IEEE Std 2003.1-1992 {4} 03

UNUSED

Assertions cited by reference assertions are typically named rather than numbered. An attempt has been made to choose meaningful names.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Because the syntactic conventions used in this standard are those of ISO/IEC 13210:1999 {5}, general assertions and general documentation assertions are rewritten in this document, even if they are also found in IEEE Std 2003.1-1992 {4}.

1.4.1.1 Phrases

The following phrases are commonly used in this standard; they have the indicated meanings.

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

This means that under the immediately preceding heading there are no new conformance requirements specified in POSIX.1b {3} that need to be tested; hence, there are no assertions. It does not mean that there are requirements for a conforming implementation that were specified in IEEE Std 1003.1b-1993 that must be satisfied. The assertions for those requirements are specified in IEEE Std 2003.1-1992 {4}.

There are no requirements for conforming implementations in this clause.

This means that there are no requirements for a conforming implementation specified in either POSIX.1b {3} or in IEEE Std 1003.1b-1993 under the immediately preceding heading.

1.4.1.2 Macros

This standard uses macros in order to save space and to make it easier to read. Macros are given long meaningful names so that the reader can know what test is covered by the assertion the macro represents. The key to reading a macro is to read and understand its definition. Macro definitions are given by the macro name followed by an equals sign (=). The following is a macro definition for an assertion that specifies the test for the proper C Standard {2} prototype being declared:

M_GA_stdC_proto_decl(func_type; func_name; param1, param2, ...; header1; header2; header3; header 4)=
IF standard THEN
SETUP: The headers `<header1>, <header2>, <header3>,` and `<header 4>` are included.
TEST: The function prototype *func_type func_name(param1, param2, ...)* is declared.
ELSE *NO_OPTION*

When a macro is used to indicate a test for a specific function, the reader should substitute the parameters in the macro call for those in the same position in the macro definition.

1.4.1.3 Cross References

Each interface definition in POSIX.1b {3} contains a Section Cross-References clause, that lists other, related parts of the standard. None of these Section Cross-References clauses contain any requirements for conforming implementations; to conserve printing expense, the corresponding sections are omitted in this standard.

1.4.2 Abbreviations

For the purposes of this standard, the following abbreviations apply:

1.4.2.1 C Standard: ISO/IEC 9899:1990, *Programming languages–C* {2}.

1.4.2.2 IRV: The International Reference Version coded character set described in ISO/IEC 646:1991 {1}.

1.4.2.3 POSIX.1b: IEEE Std 1003.1b-1993 {3}.

1.4.2.4 POSIX.1tm: IEEE Std 2003.1-1992 {4}.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

1.4.2.5 POSIX.tm: ISO/IEC 13210:1999 {5}.

1.4.2.6 POSIX.1b: POSIX Conformance Document, as specified in clause 1.3.1.2 of IEEE Std 1003.1b-1993 as amended by POSIX.1b {3}.

1.4.2.7 IUT: Implementation under test, the software that implements POSIX.1b {3}.

1.4.3 PCTS Variables

The variables that control what assertions to test and indicate which facilities an implementation provides are defined in Table 1-1.

There is at least one PCTS variable for each of the new interfaces defined by POSIX.1b {3}. The PCTS variable can be TRUE depending on one of two conditions: either the defined POSIX.1b {3} option (for example, POSIX_SEMAPHORES) is TRUE, or the implementation supports the interfaces even though it does not necessarily support all of the functionality associated with the implementation option. By combining these two conditions in the definition of the PCTS variable, the expression of conditions under which to test an assertion is made significantly easier.

Some of the interface variables contain the string "_GAP_" or "_RAP_". These strings indicate that the interface variable should be TRUE if there is a way to "get appropriate privilege" and to "release appropriate privilege," respectively.

It is recommended that a conforming test method provide a checklist, or the equivalent, to help users specify the values of the following PCTS symbols.

Table 1-1 – PCTS Symbols and Values

Symbol	Value
PCTS_AIO_CANCELABLE_OPS	**TRUE** if there are I/O operations that can be canceled by calling aio_cancel(), else **FALSE**.
PCTS_AIO_MAX	The lesser of {ARG_MAX}, as obtained from *sysconf*(), and 10 times {_POSIX_ARG_MAX}.
PCTS_APPEND_WRITE_SAME_ORDER	**TRUE** if the implementation does not relax the ordering restriction on asynchronous I/O appends occurring in the same order they were issued, else **FALSE**.
PCTS_CHMOD_SGID	**TRUE** if there are no implementation-defined restrictions that will cause the S_ISGID bit to be ignored in the *mode* argument to a *chmod*() call, else **FALSE**.
PCTS_CHMOD_SUID	**TRUE** if there are no implementation-defined restrictions that will cause the S_ISUID bit to be ignored in the mode *argument to a* chmod() call, else **FALSE**.
PCTS_DELAYTIMER_MAX	The lesser of {DELAYTIMER_MAX}, as obtained from *sysconf*(), and 8 times {POSIX_DELAYTIMER_MAX }.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Symbol	Value
PCTS_DETECT_AIO_ERROR_AIOCBP	**TRUE** if the implementation can detect that an *aiocbp* argument does not refer to an asynchronous I/O operation whose return status has not yet been retrieved.
PCTS_DETECT_ENOSPC	**TRUE** if the implementation can detect that an out-of-space condition occurs when writing to a device containing the specified file, else **FALSE**.
PCTS_DETECT_EPERM_CLOCK_SETTIME	**TRUE** if the implementation can detect that a process does not have the appropriate privileges to set the specified clock in a call to the *clock_settime* () function, else **FALSE**.
PCTS_DETECT_INVALID_FLAGS_MMAP	**TRUE** if the implementation can detect that the arguments of *addr* or *off* are not multiples of {PAGESIZE} in a call to the *mmap*() function, else **FALSE.**
PCTS_DETECT_LOCKABLE_MEMORY_LIMIT_MLOCK	**TRUE** if the implementation can detect that locking the pages mapped by the specified range would exceed an implementation-defined limit on the amount of memory that a process may lock, else **FALSE**.
PCTS_DETECT_LOCKABLE_MEMORY_LIMIT_MLOCKALL	**TRUE** if the implementation can detect that locking all of the pages currently mapped into the address space of a process would exceed an implementation-defined limit on the amount of memory that a process may lock, else **FALSE**.
PCTS_DETECT_MESSAGE_DATA_CORRUPTION	**TRUE** if the implementation can detect that a data corruption problem with a message in a message queue, else **FALSE.**
PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE	**TRUE** if the implementation can detect that the *addr* argument is not a multiple of the page size {PAGESIZE}, else **FALSE**.
PCTS_DETECT_NO_AP	**TRUE** if the implementation can detect that the calling process does not have appropriate privilege to perform the requested operation, else **FALSE**. Used by the *mlock*() and *mlockall*() functions.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Symbol	Value
PCTS_EINVAL_fchmod	**TRUE** if the implementation can detect that the *fildes* argument refers to a pipe and the implementation disallows execution of *fchmod*() on a pipe, else **FALSE.**
PCTS_EXTEND_ON_ftruncate	**TRUE** if the implementation extends a file to the *length* specified in a call to the *ftruncate*() function if the file previously was smaller that this size, else **FALSE**.
PCTS_GAP_mlock	**TRUE** if a process can get the appropriate privilege to lock process memory with *mlock*(), else **FALSE**.
PCTS_GAP_mlockall	**TRUE** if a process can get the appropriate privilege to lock process memroy with *mlockall*(), else **FALSE**.
PCTS_GAP_MODES_fchmod	**TRUE** if a process can get the appropriate privilege to change the file permission bits of a file using *fchmod*(), else **FALSE**.
PCTS_GAP_mq_open	**TRUE** if a process can get the appropriate privilege to send and receive messages in the message queue specified in *mq_open*(), else **FALSE**.
PCTS_GAP_sem_init	**TRUE** if a process can get the appropriate privilege to initialize a semaphore using *sem-init*(), else **FALSE**.
PCTS_GAP_SGID_fchmod	**TRUE** if a process can get the appropriate privilege, when the effective user ID of the calling process does not match the file owner, to change the S_ISGID file permission bits of the file using *fchmod*(), else **FALSE.**
PCTS_GAP_sigqueue	**TRUE** if a process can get the appropriate privilege to change send a signal to the receiving process using *sigqueue*(), else **FALSE.**
PCTS_GAP_SUID_fchmod	**TRUE** if a process can get the appropriate privilege, when the effective user ID of the calling process does not match the file owner, to change the S_ISUID file permission bits of the file using *fchmod*(), else **FALSE.**
PCTS_GAP_clock_settime	**TRUE** if a process can get the appropriate privilege to set a particular clock using *clock-settime*(), else **FALSE.**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Symbol	Value
PCTS_GAP_sched_setparam	**TRUE** if a process can get the appropriate privilege to set its own scheduling parameters or those of another using *sched_setparam*(), else **FALSE.**
PCTS_GAP_sched_setscheduler	**TRUE** if a process can get the appropriate privilege to change the scheduling parameters of another process or itself using *sched-setscheduler*(), else **FALSE.**
PCTS_GTI_DEVICE	**TRUE** if the implementation provides device types that support the General Terminal Interface, else **FALSE.**
PCTS_INVALID_SIGNAL	**TRUE** if the implementation has an invalid signal number, else **FALSE.**
PCTS_MAP_FIXED	**TRUE** if the implementation supports the use of the MAP_FIXED mode of memory mapping, else **FALSE.**
PCTS_MAP_PRIVATE	**TRUE** if the implementation supports the facilities indicated by the MAP_PRIVATE flat defined in the `<sys/mman.h>`, else **FALSE.**
PCTS_MORE_SA_SIGINFO_SIGNALS	**TRUE** if the implementation supports the setting of the *si_code* member of the *siginfo_t* structure by means other than calling *kill*(), *raise*(), and *abort*() (if *si_code* is set to SI_USER), *sigqueue*(), or *timer_settime*(), or by completion of an asynchronous I/O request, or by the arrival of a message on an empty message queue, else **FALSE.**
PCTS_MQ_AS_FILE_TYPE	**TRUE** if the implementation supports message queues as file type, else **FALSE.**
PCTS_MQ_OPEN_MAX	The lesser of {MP_OPEN_MAX}, as obtained from *sysconf*IO, and 256.
PCTS_MULTIPLE_OF_PAGESIZE	**TRUE** if the implementation requires that *addr* be a multiple of the page size, {PAGESIZE}, else **FALSE**.
PCTS_NAME_MAX	The lesser of {NAME_MAX}, as obtained from *pathconf*(), and 2048.
PCTS_NO_SYNC_IO_FILE	**TRUE** if the implementation has file types for which synchronized I/O is not supported, else **FALSE.**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Symbol	Value
PCTS_OPEN_MAX	The lesser of {OPEN_MAX}, as obtained from *sysconf*(), and 256.
PCTS_PATH_MAX	The lesser of {PATH_MAX}, as obtained from *sysconf*(), and 4096.
PCTS_PIPE_BUF	The lesser of {PIPE_BUF}, as obtained from *sysconf*(), and 32767.
PCTS_RAP_mlock	**TRUE** if the implementation supports releasing appropriate privilege for *mlock*(), else **FALSE.**
PCTS_RAP_mlockall	**TRUE** if the implementation supports releasing appropriate privilege for *mlockall*(), else **FALSE.**
PCTS_RAP_mq_open	**TRUE** if the implementation supports releasing appropriate privilege for *mq_open*(), else **FALSE.**
PCTS_RAP_sem_init	**TRUE** if the implementation supports releasing appropriate privileges for *sem_init*(), else **FALSE.**
PCTS_RAP_SGID_chmod	**TRUE** if the implementation supports releasing appropriate privilege for managing the S_ISGID bit in a call to *chmod*(), else **FALSE.**
PCTS_RAP_SGID_fchmod	**TRUE** if the implementation supports releasing appropriate privilege for managing the S_ISUID bit in a call to *fchmod*(), else **FALSE.**
PCTS_RAP_sigqueue	**TRUE** if the implementation supports releasing appropriate privilege for *sigqueue*(), else **FALSE.**
PCTS_RAP_clock_settime	**TRUE** if the implementation supports releasing appropriate privilege for *clock-settime*(), else **FALSE.**
PCTS_RAP_sched_setparam	**TRUE** if the implementation supports releasing appropriate privilege for *sched-setparam*(), else **FALSE.**
PCTS_ROFS	**TRUE** if the implementation supports read-only file systems, else **FALSE.**
PCTS_SEM_EBUSY	**TRUE** if the implementation supports the detection of the [EBUSY] error condition for semaphores, else **FALSE.**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Symbol	Value
PCTS_SEM_INVALID	**TRUE** if the implementation supports a way to obtain an invalid semaphore, else **FALSE.**
PCTS_SEM_IS_FD	**TRUE** if the implementation supports semaphores as a file type, else **FALSE.**
PCTS_SEM_NSEMS_MAX	The lesser of {SEM_SEMS_MAX}, as obtained from *sysconf*(), and 1024.
PCTS_SHM_AS_FILE_TYPE	**TRUE** if the implementation supports shared memory objects as a distinct file type, else **FALSE.**
PCTS_SIGQUEUE_MAX	The lesser of {SIGQUEUE_MAX}, as obtained from *sysconf*(), and 64.
PCTS_SIGTIMEDWAIT_VALUE	**TRUE** if the implementation can detect when the *sigtimedwait*() function is called with a *timeout* argument specifying a *tv_nsec* value less than zero or greater than or equal to 1000 million, else **FALSE**.
PCTS_TIMER_MAX	The lesser of {TIMER_MAX}, as obtained from *sysconf*(), and 256.
PCTS_UNSUPPORTED_SIGNAL	**TRUE** if the implementation has an unsupported signal number else **FALSE.**
PCTS_aio_cancel	**TRUE** if _POSIX_ASYNCHRONOUS_IO is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_aio_error	**TRUE** if _POSIX_ASYNCHRONOUS_IO is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_aio_fsync	**TRUE** if _POSIX_ASYNCHRONOUS_IO and _POSIX_SYNCHRONIZED_IO are defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_aio_read	**TRUE** if _POSIX_ASYNCHRONOUS_IO is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Symbol	Value
PCTS_aio_return	**TRUE** if _POSIX_ASYNCHRONOUS_IO is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_aio_suspend	**TRUE** if _POSIX_ASYNCHRONOUS_IO is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_aio_write	**TRUE** if _POSIX_ASYNCHRONOUS_IO is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_clock_getres	**TRUE** if POSIX_TIMERS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_clock_gettime	**TRUE** if _POSIX_TIMERS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_clock_settime	**TRUE** if _POSIX_TIMERS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_fchmod	**TRUE** if _POSIX_MAPPED_FILES or _POSIX_SHARED_MEMORY_OBJECTS are defined, or the implementation supports the function as described in POSIX.1b{3}, else **FALSE**.
PCTS_fdatasync	**TRUE** if _POSIX_SYNCHRONIZED_IO is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_fsync	**TRUE** if _POSIX_ASYNCHRONOUS_IO and _POSIX_SYNCHRONIZED_IO are defined or the implementation supports the function as described in POSIX.1b{3}, else **FALSE.**
PCTS_ftruncate	**TRUE** if _POSIX_MAPPED_FILES or _POSIX_SHARED_MEMORY_OBJECTS are defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_get_priority_max	**TRUE** if _POSIX_PRIORITY_SCHEDULING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Symbol	Value
PCTS_get_priority_min	**TRUE** if _POSIX_PRIORITY_SCHEDULING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_lio_listio	**TRUE** if _POSIX_ASYNCHRONOUS_IO is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_mlock	**TRUE** if _POSIX_MEMLOCK_RANGE_ is defined or the implementation supports the function as described in POSIX.1b{3}, else **FALSE**.
PCTS_mlockall	**TRUE** if _POSIX_MEMLOCK is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_mmap	**TRUE** if _POSIX_MAPPED_FILES or _POSIX_SHARED_MEMORY_OBJECTS are defined, or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_mprotect	**TRUE** if _POSIX_MEMORY_PROTECTION is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_mq_close	**TRUE** if _POSIX_MESSAGE_PASSING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_mq_getattr	**TRUE** if _POSIX_MESSAGE_PASSING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_mq_notify	**TRUE** if _POSIX_MESSAGE_PASSING and _POSIX_REALTIME_SIGNALS are defined, or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_mq_open	**TRUE** if _POSIX_MESSAGE_PASSING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Symbol	Value
PCTS_mq_receive	**TRUE** if _POSIX_MESSAGE_PASSING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_mq_send	**TRUE** if _POSIX_MESSAGE_PASSING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_mq_setattr	**TRUE** if _POSIX_MESSAGE_PASSING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_mq_unlink	**TRUE** if _POSIX_MESSAGE_PASSING is defined or the implementation supports the function as described in POSIX.1b {3], else **FALSE.**
PCTS_msync	**TRUE** if _POSIX_MAPPED_FILES and _POSIX_SYNCHRONIZED_IO are defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_msync_storage	**TRUE** if the system under test has secondary storage to which *msync*() can synchronize pages, else **FALSE**.
PCTS_munlock	**TRUE** if _POSIX_MEMLOCK_RANGE is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_munlockall	**TRUE** if _POSIX_MEMLOCK is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_munmap	**TRUE** if _POSIX_MAPPED_FILES or _POSIX_SHARED_MEMORY_OBJECTS are defined, or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_nanosleep	**TRUE** if _POSIX_TIMERS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_read	**TRUE** if the implementation supports the *read*() function (always **TRUE**).

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Symbol	Value
PCTS_sched_get_priority_max	**TRUE** if _POSIX_PRIORITY_SCHEDULING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sched_get_priority_min	**TRUE** if _POSIX_PRIORITY_SCHEDULING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sched_getparam	**TRUE** if _POSIX_PRIORITY_SCHEDULING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sched_getscheduler	**TRUE** if _POSIX_PRIORITY_SCHEDULING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sched_rr_get_interval	**TRUE** if _POSIX_PRIORITY_SCHEDULING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sched_setparam	**TRUE** if _POSIX_PRIORITY_SCHEDULING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sched_setscheduler	**TRUE** if _POSIX_PRIORITY_SCHEDULING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sched_yield	**TRUE** if _POSIX_PRIORITY_SCHEDULING is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sem_close	**TRUE** if _POSIX_SEMAPHORES is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sem_destroy	**TRUE** if _POSIX_SEMAPHORES is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Symbol	Value
PCTS_sem_getvalue	**TRUE** if _POSIX_SEMAPHORES is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sem_init	**TRUE** if _POSIX_SEMAPHORES is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sem_open	**TRUE** if _POSIX_SEMAPHORES is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sem_post	**TRUE** if _POSIX_SEMAPHORES is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sem_trywait	**TRUE** if _POSIX_SEMAPHORES is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_sem_unlink	**TRUE** if _POSIX_SEMAPHORES is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sem_wait	**TRUE** if _POSIX_SEMAPHORES is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE**.
PCTS_shm_open	**TRUE** if _POSIX_SHARED_MEMORY _OBJECTS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_shm_unlink	**TRUE** if _POSIX_SHARED_MEMORY _OBJECTS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sigqueue	**TRUE** if _POSIX_REALTIME_SIGNALS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_sigtimedwait	**TRUE** if _POSIX_REALTIME_SIGNALS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Symbol	Value
PCTS_sigwaitinfo	**TRUE** if _POSIX_REALTIME_SIGNALS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_timer_create	**TRUE** if _POSIX_TIMERS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_timer_delete	**TRUE** if _POSIX_TIMERS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_timer_getoverrun	**TRUE** if _POSIX_TIMERS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_timer_gettime	**TRUE** if _POSIX_TIMERS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_timer_settime	**TRUE** if _POSIX_TIMERS is defined or the implementation supports the function as described in POSIX.1b {3}, else **FALSE.**
PCTS_write	**TRUE** if the implementation supports the *write* () function. (always **TRUE**).

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 2: Terminology and General Requirements

2.1 Conventions

This standard uses the following typographic conventions:

(1) The *italic* font is used for

-- Cross references to defined terms within 1.3, 2.2.1, and 2.2.2, symbolic parameters that are generally substituted with real values by the application

-- C language data types and function names (except in function Synopsis clauses)

-- Global external variable names

-- General Assertion and General Documentation Assertion references.

(2) The **bold** font is used with a word in all capital letters, such as **PATH**, to represent an environmental variable. It is also used for the term "**NULL** pointer."

(3) The `constant-width` (Courier) font is used

- For C language data types and function names within function Synopsis clauses

- To illustrate examples of system input or output where exact usage is depicted

- For references to utility names and C language headers.

(4) Symbolic constants returned by many functions as error numbers are represented as

[ERRNO]

(5) Symbolic constants or limits defined in certain headers are represented as

[LIMIT]

(6) Test method macros are represented as **M_test_method_macro ().**

In some cases, tabular information is presented "inline"; in other cases it is presented in a separately labeled table. This arrangement was used purely for ease of typesetting, and there is no normative difference between these two cases.

The conventions listed above are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in this standard.

NOTEs provided as parts of labeled tables and figures are integral parts of this standard (normative). Footnotes and notes within the body of the text are for information only (informative).

Numerical quantities are presented in international style; that is, the comma is used as a decimal sign, and units are from the International System (SI).

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.2 Definitions

2.2.1 Terminology

There are no requirements for conforming implementations in this clause.

2.2.2 General Terms

There are no requirements for conforming implementations in this clause.

2.2.2.1 absolute pathname: There are no requirements for conforming implementations in this clause.

2.2.2.2 access mode: There are no requirements for conforming implementations in this clause.

2.2.2.3 address space: There are no requirements for conforming implementations in this clause.

2.2.2.4 appropriate privileges: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.5 arm (a timer): There are no requirements for conforming implementations in this clause.

2.2.2.6 asynchronous I/O completion: There are no requirements for conforming implementations in this clause.

2.2.2.7 asynchronous I/O operation: There are no requirements for conforming implementations in this clause.

2.2.2.8 background process: There are no requirements for conforming implementations in this clause.

2.2.2.9 background process group: There are no requirements for conforming implementations in this clause.

2.2.2.10 block special file: There are no requirements for conforming implementations in this clause.

2.2.2.11 blocked process: There are no requirements for conforming implementations in this clause.

2.2.2.12 character: There are no requirements for conforming implementations in this clause.

2.2.2.13 character special file: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.14 child process: There are no requirements for conforming implementations in this clause.

2.2.2.15 clock: There are no requirements for conforming implementations in this clause.

2.2.2.16 clock tick: There are no requirements for conforming implementations in this clause.

2.2.2.17 controlling process: There are no requirements for conforming implementations in this clause.

2.2.2.18 controlling terminal: There are no requirements for conforming implementations in this clause.

2.2.2.19 current working directory: There are no requirements for conforming implementations in this clause.

2.2.2.20 device: There are no requirements for conforming implementations in this clause.

2.2.2.21 directory: There are no requirements for conforming implementations in this clause.

2.2.2.22 direct I/O: There are no requirements for conforming implementations in this clause.

2.2.2.23 directory entry [link]: There are no requirements for conforming implementations in this clause.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.2.2.24 disarm (a timer): There are no requirements for conforming implementations in this clause.

2.2.2.25 drift rate (of a clock): There are no requirements for conforming implementations in this clause.

2.2.2.26 dot: There are no requirements for conforming implementations in this clause.

2.2.2.27 dot-dot: There are no requirements for conforming implementations in this clause.

2.2.2.28 effective group ID: There are no requirements for conforming implementations in this clause.

2.2.2.29 effective user ID: There are no requirements for conforming implementations in this clause.

2.2.2.30 empty directory: There are no requirements for conforming implementations in this clause.

2.2.2.31 empty string [null string]: There are no requirements for conforming implementations in this clause.

2.2.2.32 Epoch: There are no requirements for conforming implementations in this clause.

2.2.2.33 feature test macro: There are no requirements for conforming implementations in this clause.

2.2.2.34 FIFO special file [FIFO]: There are no requirements for conforming implementations in this clause.

2.2.2.35 file: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.36 file description: There are no requirements for conforming implementations in this clause.

2.2.2.37 file descriptor: There are no requirements for conforming implementations in this clause.

2.2.2.38 file group class: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.39 file mode: There are no requirements for conforming implementations in this clause.

2.2.2.40 filename:
(IEEE Std 2003.1-1992 {4} GA02
UNUSED

M_GA_*portableFilenames*(*function*()) =
TEST: **The interface *function ()*** supports filenames containing any of the characters in the portable filename character set.
TR: Test for filenames containing the following characters:
A B D C E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -
The last three characters are the period, underscore, and hyphen, respectively.

GA_portableFilenames
FOR: *execl(), execle(), execv(), execve(), exelp(), execvp(), opendir(), chdir(), open(), creat(), link(), existing, link() new, mkdir(), mkfifo(), unlink(), rmdir(), rename(), old, rename(), new, stat(), access(), chmod(), chown(), utime(), pathconf(), fopen(), freopen(), remove(),* `tar` format creating utility, and `cpio` format creating utility.
M_GA_portableFilenames(function())
Conformance for definition: PASS

(IEEE Std 2003.1-1992 {4} GA03
UNUSED

M_GA_upperLowerNames*(function())* =
TEST: The interface *function*() differentiates between uppercase and lowercase characters in filenames.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

GA_upperLowerNames

FOR: *execl(), execle(), execv(), execve(), execlp(), execvp(), opendir(), chdir(), open(), creat(), link() existing, link() new, mkdir(), mkfifo(), unlink(), rmdir(), rename() old, rename() new, stat(), access(), chmod(), chown(), utime(), pathconf(), fopen(), freopen(), remove(),* `tar` format creating utility, and `cpio` format creating utility.

M_GA_upperLowerNames*(function())*

Conformance for definitions: PASS

2.2.2.41 file offset: There are no requirements for conforming implementations in this clause.

2.2.2.42 file other class: There are no requirements for conforming implementations in this clause.

2.2.2.43 file owner class: There are no requirements for conforming implementations in this clause.

2.2.2.44 file permission bits: There are no requirements for conforming implementations in this clause.

2.2.2.45 file serial number: There are no requirements for conforming implementations in this clause.

2.2.2.46 file system: There are no requirements for conforming implementations in this clause.

2.2.2.47 first open (of a file): `cpio` format creating utility. There are no requirements for conforming implementations in this clause.

2.2.2.48 foreground process: There are no requirements for conforming implementations in this clause.

2.2.2.49 foreground process group: There are no requirements for conforming implementations in this clause.

2.2.2.50 foreground process group ID: There are no requirements for conforming implementations in this clause.

2.2.2.51 group ID: There are no requirements for conforming implementations in this clause.

2.2.2.52 job control: There are no requirements for conforming implementations in this clause.

2.2.2.53 last close (of a file) There are no requirements for conforming implementations in this clause.

2.2.2.54 link: There are no requirements for conforming implementations in this clause.

2.2.2.55 link count: There are no requirements for conforming implementations in this clause.

2.2.2.56 login: There are no requirements for conforming implementations in this clause.

2.2.2.57 login name: There are no requirements for conforming implementations in this clause.

2.2.2.58 map: There are no requirements for conforming implementations in this clause.

2.2.2.59 memory object: There are no requirements for conforming implementations in this clause.

2.2.2.60 memory-resident: There are no requirements for conforming implementations in this clause.

2.2.2.61 message: There are no requirements for conforming implementations in this clause.

2.2.2.62 message queue: There are no requirements for conforming implementations in this clause.

2.2.2.63 mode: There are no requirements for conforming implementations in this clause.

2.2.2.64 null string: There are no requirements for conforming implementations in this clause.

2.2.2.65 open file: There are no requirements for conforming implementations in this clause.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.2.2.66 open file description: There are no requirements for conforming implementations in this clause.

2.2.2.67 orphaned process group: There are no requirements for conforming implementations in this clause.

2.2.2.68 page: There are no requirements for conforming implementations in this clause.

2.2.2.69 parent directory: There are no requirements for conforming implementations in this clause.

2.2.2.70 parent process: There are no requirements for conforming implementations in this clause.

2.2.2.71 parent process ID: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.72 path prefix: There are no requirements for conforming implementations in this clause.

2.2.2.73 pathname: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.74 pathname component: There are no requirements for conforming implementations in this clause.

2.2.2.75 persistence: There are no requirements for conforming implementations in this clause.

2.2.2.76 pipe: There are no requirements for conforming implementations in this clause.

2.2.2.77 portable filename character set: There are no requirements for conforming implementations in this clause.

2.2.2.78 preallocation: There are no requirements for conforming implementations in this clause.

2.2.2.79 preempted process: There are no requirements for conforming implementations in this clause.

2.2.2.80 priority: There are no requirements for conforming implementations in this clause.

2.2.2.81 priority-based scheduling: There are no requirements for conforming implementations in this clause.

2.2.2.82 privilege: There are no requirements for conforming implementations in this clause.

2.2.2.83 process: There are no requirements for conforming implementations in this clause.

2.2.2.84 process group: There are no requirements for conforming implementations in this clause.

2.2.2.85 process group ID: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.86 process group leader: There are no requirements for conforming implementations in this clause.

2.2.2.87 process group lifetime: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.88 process ID: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.89 process lifetime: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.90 process list: There are no requirements for conforming implementations in this clause.

2.2.2.91 read-only file system: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.92 real group ID: There are no requirements for conforming implementations in this clause.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.2.2.93 real user ID: There are no requirements for conforming implementations in this clause.

2.2.2.94 referenced shared memory object: There are no requirements for conforming implementations in this clause.

2.2.2.95 regions: There are no requirements for conforming implementations in this clause.

2.2.2.96 regular file: There are no requirements for conforming implementations in this clause.

2.2.2.97 relative pathname: There are no requirements for conforming implementations in this clause.

2.2.2.98 resolution (time): There are no requirements for conforming implementations in this clause.

2.2.2.99 root directory: There are no requirements for conforming implementations in this clause.

2.2.2.100 runnable process: There are no requirements for conforming implementations in this clause.

2.2.2.101 running process: There are no requirements for conforming implementations in this clause.

2.2.2.102 saved set-group-ID: There are no requirements for conforming implementations in this clause.

2.2.2.103 saved set-user-ID: There are no requirements for conforming implementations in this clause.

2.2.2.104 scheduling: There are no requirements for conforming implementations in this clause.

2.2.2.105 scheduling policy:

D_1 TEST: The PCD.1b shall define in clause 2.2.2.105 the manner in which each of the scheduling policies may modify the priorities or otherwise affect the ordering of processes at each of the following occurrences

(1) When a process is a running process and it becomes a blocked process

(2) When a process is a running process and it becomes a preempted process

(3) When a process is a blocked process and it becomes a runnable process

(4) When a running process calls a function that can change the priority or scheduling policy of a process

(5) In other scheduling-policy-defined circumstances.

Conformance for definitions: PASS

D_2 TEST: The PCD.1b defines in clause 2.2.2.105 under what other circumstances and in what manner each scheduling policy may modify the priorities or otherwise affect the ordering of processes.

Conformance for definitions: PASS

2.2.2.106 seconds since the Epoch: There are no requirements for conforming implementations in this clause.

2.2.2.107 semaphore: There are no requirements for conforming implementations in this clause.

2.2.2.108 semaphore lock operation:

R_1 FOR: *sem_init()* and *sem_open()*
IF *PCTS_sem_wait* ***THEN***
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function* ().

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: When a call to *sem_wait (sem)* completes successfully, the interface returns a value of 0, and the semaphore designated by *sem* is locked by the semaphore lock operation.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion sem_wait in 11.2.7.2

2.2.2.109 semaphore unlock operation:

R_2 FOR: *sem_init()* and *sem_open()*

IF *PCTS_sem_post* ***THEN***

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function* ().

TEST: A successful call to *sem_post*() unlocks the semaphore referenced by *sem* by performing the semaphore unlock operation on that semaphore, and returns the value zero.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion sem_post in 11.2.7.2

2.2.2.110 session: There are no requirements for conforming implementations in this clause.

2.2.2.111 session leader: There are no requirements for conforming implementations in this clause.

2.2.2.112 session lifetime: There are no requirements for conforming implementations in this clause.

2.2.2.113 shared memory object: There are no requirements for conforming implementations in this clause.

2.2.2.114 signal: There are no requirements for conforming implementations in this clause.

2.2.2.115 slash: There are no requirements for conforming implementations in this clause.

2.2.2.116 supplementary group ID: There are only IDDD Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.2.2.117 successfully transferred: There are no requirements for conforming implementations in this clause.

2.2.2.118 synchronized I/O completion: There are no requirements for conforming implementations in this clause.

2.2.2.119 synchronized I/O data integrity completion:

GA_syncIODataIntegrityRead

FOR: *read(), aio_read(),* and *lio_listio()*

IF *PCTS_function* and {POSIX_SYNCH_IO} **THEN**

SETUP: Open a file by calling *open*() with O_RSYNC and O_DSYNC set in the *oflag* parameter.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: At the time that the synchronized read operation initiated by calling *function*() occurs, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data.

TR: Test for regular files.

ELSE *NO_OPTION*

Conformance for definitions: PASS, NO_TEST, NO_OPTION

GA_syncIODataIntegrityWbeforeR

FOR: *write(), aio_write(),* and *lio_liosio()*

IF *PCTS_function* and {POSIX_SYNCH_IO} **THEN**

SETUP: Open a file by calling *open()* with O_DSYNC set in the *oflag* parameter.

TEST: A write operation initiated by calling *function*() either completes by transferring an image of the data to the physical medium containing the file or, if unsuccessful, by diagnosing and returning an indicator of the error.

TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.

ELSE *NO_OPTION*

Conformance for definitions: PASS, NO_TEST, NO_OPTION

GA_syncIODataIntegrityWrite

FOR: *write*(), *aio_write*(), and *lio_listio*()

IF *PCTS_funtion and* {POSIX_SYNCH_IO} **THEN**

SETUP: Open a file by calling *open*() with O_DSYNC set in the *oflag* parameter.

TEST: A write operation initiated by calling *function*() either completes by transferring an image of the date to the physical medium containing the file or, if unsuccessful, by diagnosing and returning an indicator of the error.

TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.

ELSE *NO_OPTION*

Conformance for definitions: PASS, NO_TEST, NO_OPTION

2.2.2.120 synchronized I/O file integrity completion:

GA_syncIOFileIntegrityRead

FOR: *read(), aio_read(),* and *lio_listio()*

IF *PCTS_function* and {POSIX_SYNCH_IO} **THEN**

SETUP Open a file by calling *open()* with O_RSYNC and O_SYNC set in the *oflag* parameter.

TEST: At the time that the synchronized read operation initiated by calling *function()* occurs, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data, and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode
2. File serial number
3. ID of device containing this file
4. Number of links
5. User ID of the owner of the file
6. Group ID of the group of the file
7. The file size in bytes
8. Time of last access

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9. Time of last data modification

10. Time of last file status change

TR: Test for regular files.

ELSE *NO_OPTION*

Conformance for definitions: PASS, NO_TEST, NO_OPTION

GA_syncIOFileIntegrityWrite

FOR: write(), aio_write(), *and* lio_listio()

IF *PCTS_function* and {POSIX_SYNCH_IO} **THEN**

TEST: At the time that the synchronized write operation initiated by calling *function*() occurs, the data are written to the physical medium containing the file, and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode

2. File serial number

3. ID of device containing this file

4. Number of links

5. User ID of the owner of the file

6. Group ID of the group of the file

7. The file size in bytes

8. Time of last access

9. Time of last data modification

10. Time of last file status change

TR: Test for regular files

ELSE *NO_OPTION*

Conformance for definitions: PASS, NO_TEST, NO_OPTION

2.2.2.121 synchronized I/O operation: There are no requirements for conforming implementations in this clause.

2.2.2.122 synchronous I/O operation: There are no requirements for conforming implementations in this clause.

2.2.2.123 system: There are no requirements for conforming implementations in this clause.

2.2.2.124 system crash: There are no requirements for conforming implementations in this clause.

2.2.2.125 system process: There are no requirements for conforming implementations in this clause.

2.2.2.126 system reboot:

D_3 TEST: The PCD.1b defines in clause 2.2.2.126 the implementation-defined sequence of events (called a system reboot) that may result in the loss of transitory data; i.e., data that is not saved in permanent storage.

Conformance for definitions: PASS

2.2.2.127 terminal [terminal device]: There are no requirements for conforming implementations in this clause.

2.2.2.128 timer: There are no requirements for conforming implementations in this clause.

2.2.2.129 timer overrun: There are no requirements for conforming implementations in this clause.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.2.2.130 user ID: There are no requirements for conforming implementations in this clause.

2.2.2.131 user name: There are no requirements for conforming implementations in this clause.

2.2.2.132 working directory [current working directory]: There are no requirements for conforming implementations in this clause.

2.2.3 Abbreviations

There are no requirements for conforming implementations in this clause.

See clause 1.4.2 for abbreviations related to this standard.

2.3 General Concepts

2.3.1 extended security controls: There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.3.2 file access permissions:
(IEEE Std 2003.1-1992 {4} GA04
UNUSED

M_GA_AP_overrideFileAccess*(function()) =*
IF the IUT provides a mechanism for creating processes with the appropriate privilege to override a file access control mechanism **THEN**
SETUP: The process has appropriate privileges for file access.
TEST: A call to the interface *function*() that needs read, write, or search access to the *pathname* argument is granted access to the file when access would otherwise be denied.
ELSE *NO_OPTION*

GA_AP_overrideFileAccess
FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execlp(), execvp(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), rename() new, rename() old, rmdir(), stat(), unlink(),* and *utime().*
M_GA_AP_overrideFileAccess*(function())*
Conformance for General Concepts; PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4} GA05
UNUSED

M_GA_AP_overrideExecAccess*(function()) =*
IF the IUT provides a mechanism for creating processes with the appropriate privilege to override a file access control mechanism **THEN**
SETUP: The process has appropriate privilege for the file access, and execute permission is granted to at least one user of the file.
TEST: A call to the interface *function*() that needs execute permission to the *path* or *file* argument is granted execute access to the file when access would otherwise be denied.
ELSE *NO_OPTION*

GA_AP_overrideExecAccess
FOR: *execl(), execle(), execv(), execve(), execlp(),* and *execvp().*
M_GA_AP_overrideExecAccess*(function())*
Conformance for General Concepts: PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4} GA06
UNUSED

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

***M_GA_AP_classAccess**(function()) =*

IF the IUT provides a mechanism for creating processes with the appropriate privilege to override a file access control mechanism **THEN**

SETUP: The process does not have appropriate privilege to override the file access control mechanism, and the process requires read, write, execute, or search access to the *pathname* or *file* argument of the interface *function*().

TEST: A call to the interface *function*() is granted access to the file when the required access permission bit is set for the class (file owner class, file group class, or file other class) to which the process belongs.

ELSE *NO_OPTION*

GA_AP_classAccess

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execlp(), execvp(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), rename() new, rename() old, rmdir(), stat(), unlink(),* and *utime().*

***M_GA_AP_ClassAccess**(function())*

Conformance for General Concepts: PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4} GA07

UNUSED

***M_GA_AdditionalAccessControl**()) =*

IF IUT provides additional file access control mechanisms **THEN**

TEST: Any additional file access control mechanism shall only further restrict the access permissions defined by the file permission bits.

ELSE *NO_OPTION*

GA_AdditionalAccessControl

MP_GA_AdditionalAccessControl()

Conformance for General Concepts: PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4} GA08

UNUSED

MP_GA_AlternateAccessControl() =

IF IUT provides alternate file access control mechanisms **THEN**

TEST: Any alternate file access control mechanism specifies file permission bits for the file owner class, file group class, and file other class of the file corresponding to the access permissions to be returned by *stat*() and *fstat()*.

ELSE *NO_OPTION*

GA_AlternateAccessControl

M_GA_AlternateAccessControl()

Conformance for General Concepts, PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4}) GA09

UNUSED

(IEEE Std 2003.1-1992 {4}) GA10

UNUSED

M_GA_AltAccessEnable () =

IF IUT provides alternate file access control mechanisms **THEN**

TEST: The alternate file access control mechanisms can be enabled only by explicit user action, on a per-file basis, by the file owner or a user with the appropriate privilege.

ELSE *NO_OPTION*

GA_AltAccessEnable

M_GA_AltAccessEnable()

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for General Concepts: PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4}) GA11
UNUSED

M_GA_AltAccessDisable() =
IF IUT provides alternate file access control mechanisms **THEN**
TEST: The alternate file access control mechanisms will be disabled for a file after the file permission bits are changed for that file with *chmod*().
ELSE *NO_OPTION*

GA_AltAccessDisable
M_GA_AltAccessDisable()
Conformance for General Concepts: PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4}) D03
UNUSED

2.3.3 file hierarchy: There are no requirements for conforming implementations in this clause.

2.3.4 filename portability: There are no requirements for conforming implementations in this clause.

2.3.5 file times update:
(IEEE Std 2003.1-1992 {4})R01
UNUSED

(IEEE Std 2003.1-1992 {4}) GA12
UNUSED

M_GA_StatTimeUpdate *(function())* =
TEST: The interface *function*(), when called, updates all time-related fields marked for update and does not update any time-related fields not marked for update.

GA_StatTimeUpdate
FOR: *stat()* and *fstat()*.
M_GA_StatTimeUpdate*(function())*
Conformance for General Concepts: PASS

M_GA_NoOpenTimeUpdate*(function())* =
TEST: All fields that are marked for update are updated when the file is no longer open by any process.

GA_NoOpenTimeUpdate
FOR: *close()* and *fclose*().
M_GA_NoOpenTimeUpdate *(function())*
Conformance for General Concepts: PASS

NOTE: This assertion is missing in 2003.1.

(IEEE Std 2003.1-1992 {4}_GA13
UNUSED

M_GA_NoROFSTimeUpdate*(function())* =
TEST: Time-related field updates are not done for files on read-only file systems.

GA_NoROFSTimeUpdate
FOR: *acc(), chmod(), chown(), creat(), link() existing, link() new, mkdir(), mkfifo(), open(), rename() new, rename() old, rmdir(), unlink(),* and *utime()*.
M_GA_noROFSTimeUpdate*(function())*

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for General Concepts: PASS

2.3.6 pathname resolution:

NOTE: In each of the pathname resolution General Assertions below, for the elements *rmdir(), rename()* new, and *unlink(),* the current working directory should be an empty directory in order to preclude the occurrence of avoidable error conditions. Some implementations will consider the attempt to remove the current working directory an error, and will indicate this with the error indication.

(IEEE Std 2003.1-1992 {4}) GA14

UNUSED

M_GA_PRDot*(function()) =*

TEST: A call to the interface *function()* with a *path* or *file* argument where the first filename component is "." and the argument does not begin with a "/". [slash resolves the *path* or *file* argument by locating the second filename component (when specified) in the current working directory.]

GA_PRDot

FOR: *access(), chdir(), chmod(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execlp(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink()* and *utime().*

M_GA_PRDot*(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}_ GA15

UNUSED

M_GA_PRSlash*(function()) =*

TEST: A call to the interface *function()*, with a *path* or *file* argument pointing to the string "/", resolves the *path* or *file* argument to the root directory of the process.

GA_PRSlash

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execp(), execpt(), fopen(), freopen(), open(), opendir(), oink() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(),* and *utime().*

M_GA_PRSlash*(function())*

Conformance for General Concepts: pass

(IEEE Std 2003.1-1992 {4}) GA16

UNUSED

M_GA_PR3Slash*(function()) =*

TEST: A call to the interface *function*(), with a *path* or *file* argument pointing to the string "///", resolves the *path* or *file* argument to the root directory of the process.

GA_PR3Slash

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execlp(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(),* and *utime().*

M_GA_PR3Slash*lfunction())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}) GA17

UNUSED

M_GA_PRSlashsPath*(function()) =*

TEST: A call to the interface *function(),* with a *path* or *file* argument pointing to a string that starts with either a single slash ("/") or three or more slashes, resolves the *path* or *file* argument

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

by locating the first filename component of the argument in the root directory of the process.

GA_PRSlashsPath

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execlp(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(),* and *utime().*

M_GA_PRSlashsPath*(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}) GA18

UNUSED

M_GA_PRDotDot*(function()) =*

TEST: A call to the interface *function() (),* with a *path* or *file* argument, where the first filename component is "..", the argument does not begin with a "/" (slash), and the current working directory is not the root directory of the process, resolves the *path* or *file* argument by locating the second filename component (when specified) in the parent directory of the current working directory.

GA_PRDotDot

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execle(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*

M_GA_PRDotDot*(function())*

Conformance for General Concepts; PASS

(IEEE Std 2003.1-1992 {4}) GA19

UNUSED

M_GA_PRRelativeSlash*(function()) =*

TEST: A call to the interface *function() (),* with a *path* or *file* argument pointing to the string "F1/", and where "F1" is a directory, resolves the *path* or *file* argument by locating F1 "F1" in the current working directory.

GA_PRRelativeSlash

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execle(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*

M_GA_PRRelativeSlash*(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4} GA20

UNUSED

M_GA_PRRelativeSlashSlash*(function()) =*

TEST: A call to the interface *function* () () *with a path or file argument pointing to the string* "F1//" and "F1" is a directory resolves the *path* or *file* argument by locating "F1" in the current working directory.

GA_PRRelativeSlashSlash

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execle(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*

M_GA_PRRelativeSlashSlash*(function())*

Conformance for General Concepts: PASS

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

***M_GA_PRRenameRelativeSlashSlash**(function()) =*

TEST: A call to the interface *function() (),* with a *new* argument pointing to the string "F1//", and where "F1" is an empty directory, resolves the *new* argument by locating "F1" in the current working directory.

GA_PRRenameRelativeSlashSlash

FOR: *rename()new*

***M_GA_PRRenameRelativeSlashSlash**(function())*

Conformance for general Concepts: PASS

(IEEE Std 2003.1-1992 {4})GA21

UNUSED

***M_GA_PRRelativeCWD**(function()) =*

TEST: A call to the interface *function() (),* with a *path* or *file* argument pointing to the string "F1/F2", resolves the *path* or *file* argument by locating "F2" in the directory "F1" in the current working directory.

GA_PRRelativeCWD

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execle(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*

***M_GA_PRRelativeCWD**(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}_ GA22

UNUSED

***M_GA_PRRelativeDotCWD**(function()) =*

TEST: A call to the interface *function() (),* with a *path* or *file* argument pointing to the string "F1/./F2", resolves the *path* or *file* argument by locating "F2" in the directory "F1" in the current working directory.

GA_PRRelativeDotCWD

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execle(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*

***M_GA_PRRelativeDotCWD**(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4} GA23

UNUSED

***M_GA_PRRelativeDotDotCWD**(function()) =*

TEST: A call to the interface *function() (),* with a *path* or *file* argument pointing to the string "F1/../F1/F2", resolves the *path* or *file* argument by locating "F2" in the directory "F1" in the current working directory.

GA_PRRelativeDotDotCWD

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execle(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*

***M_GA_PRRelativeDotDotCWD**(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}) GA24

UNUSED

***M_GA_PRRelativeSlashSlashCWD**(function()) =*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: A call to the interface *function() ()*, with a *path* or *file* argument pointing to the string "F1//F2", resolves the *path* or *file* argument by locating "F2" in the directory "F1" in the current working directory.

GA_PRRelativeSlashSlashCWD

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execle(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*

***M_GA_PRRelativeSlashSlashCWD**(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}) GA25

UNUSED

***M_GA_PPRnoTrunc**(function()) =*

IF {POSIX_NO_TRUNC} is not supported in the specified directory **THEN**

TEST: A call to the interface *function() ()*, with a *path* or *file* argument that has a pathname component of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, resolves the pathname component by truncating it to {NAME_MAX} bytes.

ELSE *NO_OPTION*

GA_PRNoTrunc

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execle(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*

***M_GA_PRNoTrunc**(function())*

Conformance for General Concepts: PASS, NO_OPTION

***M_GA_PRnoTruncError**(function()) =*

IF {POSIX_NO_TRUNC is supported in the specified directory **THEN**

TEST: A call to the interface *function() ()*, with a *path* or *file* argument that has a pathname component of more than {NAME_MAX} bytes in a directory for which {POSIX_NO_TRUNC} is supported, generates an [ENAMETOOLONG] error.

ELSE *NO_OPTION*

GA_PRNoTruncError

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execle(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*

***M_GA_PRNoTruncError**(function())*

Conformance for General Concepts: PASS, NO_OPTION

2.4 Error Numbers

(IEEE Std 2003.1-1992 P{4}) 02

UNUSED

2 **SETUP:** Include the header `<errno.h>`.

TEST: The error numbers[E2BIG], [EACCESS], [EAGAIN[, [EBADF], EBADMSG], EBUSY],]ECANCELED], [ECHILD], [EDEADLK], [EDOM], [EEXIST], [EFAULT[, [EFBIG], [EINPROGRESS], [EINTR], [EINTR], [EINVAL], [EIO], [EISDIR], [EMFILE], [EMLINK], [EMSGSIZE], [ENAMETOOLONG], [ENFILE], [ENODEV], [ENOENT], [ENOEXEC], [EN0LOCK], [ENOMEM], [ENOSPC], [ENOSYS], [ENOTDIR], [ENOTEMPTY], [ENOTTY], [ENXIO], [EPERM], [EPIPE], [ERANGE], [EROFS], [ESPIPE], [ESRCH], and [EXDEV] are defined, are nonzero, are distinct from each other, and can be represented in *errno*.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for Error numbers: PASS

(IEEE Std 2003.1-1992 {4} DGA02
UNUSED

***M_GD_OptionalErrors**(function()) =*
IF the IUT supports the detection of an optional error condition **THEN**
TEST: The PCD.1b contains the details of the optional error conditions detected in the clause of the PCD.1b where the error values of the interface *function*() are described.
ELSE *NO_OPTION*

GD_OptionalErrors
FOR: *access(), chown(), chosedir(), execl(), execle(), execv(), execve(), execlp(), execvp(), fcntl(), fork(), fpathconf(), getcwd(), opendir(), pathconf(), readdir(), sigaddset(), sigdelset(),* and *(sigismember().*
***M_GD_OptionalErrors**(function())*
Conformance for Error Numbers: PASS, NO_OPTION

(IEEE Std 2003.1-1992 {4}) GA26
UNUSED

***M_GA_OptionalErrorsUndetected**(function()) =*
IF the IUT does not support the detection of an optional error condition **THEN**
TEST: The action specified by a call to the interface *function*() that would otherwise generate an optional error condition will succeed.
ELSE *NO_OPTION*

GA_OptionalErrorsUndetected
FOR: *access(), chown(), chosedir(), execl(), execle(), execv(), execve(), execlp(), execvp(), fcntl(), fork(), fpathconf(), getcwd(), opendir(), pathconf(), readdir(), sigaddset(), sigdelset(),* and *(sigismember().*
***M_GA_OptionalErrorsUndetected**(function())*
Conformance for Error Numbers: PASS, NO_OPTION

2.5 Primitive System Data Types

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.6 Environment Description

(IEEE Std 2003.1-1992 {4}) GA27
UNUSED

***M_GA_ExecNOSlash ()** =*
TEST: The interfaces *execlp()* and *execvp()* use the path prefixes in the **PATH** environment variable only when their *file* argument does not contain a slash.

GA_ExecNoSlash
FOR: *execlp()* and *execvp().*
M_GA_ExecNoSlash()
Conformance for Environment Description: PASS

(IEEE Std 2003.1-1992 {4}) GA28
UNUSED

***M_GA_ExecColon()** =*
TEST: The search path used by the interfaces *execlp()* and *execvp()* uses the path prefixes in the **PATH** environment variable that are separated by a colon.

GA_ExecColon

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

FOR: *execlp()* and *execvp().*
M_GA_ExecColon()
Conformance for Environment Description: PASS

(IEEE Std 2003.1-1992 {4}) GA29
UNUSED

M_GA_ExecInsertSlash() =
TEST: The interfaces *execlp*() and *execvp()* insert a "/" between a nonzero-length path prefix in the **PATH** environment variable and the filename in the file argument when searching for an executable file.

GA_ExecInsertSlash
FOR: *execlp()* and *execvp().*
M_GA_ExecInsertSlash()
Conformance for environment Description: PASS

(IEEE Std 2003.1-1992 {4} GA30
UNUSED

M_GA_execTwoColons() =
TEST: The search path used by the interfaces *execlp()* and *execvp()* uses the current working directory as the path prefix corresponding to two adjacent colons, "::", in the **PATH** environment variable.

GA_ExecTwoColons
FOR: *execlp()* and *execvp().*
M_GA_ExecTwoColons()
Conformance for Environment Description: PASS

(IEEE Std 2003.1-1992 {4}) GA31
UNUSED

M_GA_ExecInitialColon() =
TEST: The search path used by the interfaces *execlp()* and *execvp()* uses the current working directory as the path prefix when the value of the **PATH** environment variable starts with a ":"

GA_ExecInitialColon
FOR: *execlp()* and *execvp().*
M_GA_ExecInitialColon()
Conformance for Environment Description: PASS

(IEEE Std 2003.1-1992 {4}) GA32
UNUSED

M_GA_ExecTrailingColon() =
TEST: The search path used by the interfaces *execlp()* and *execvp()* uses the current working directory as the path prefix when the value of the **PATH** environment variable ends with a ":"

GA_ExecTrailingColon
FOR: *execlp()* and *execvp().*
M_GA_ExecTrailingColon()
Conformance for Environment Description: PASS

(IEEE Std 2003.1-1992 {4}) GA33
UNUSED

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

M_GA_ExecPathSearchOrder() =

TEST: The interfaces *execlp()* and *execvp()* search the path prefixes in the **PATH** environment variable from the beginning to the end until an executable program by the specified name is found.

GA_ExecPathSearchOrder

FOR: *execlp()* and *execvp()*
M_GA_ExecPathSearchOrder()
Conformance for Environment Description: PASS

(IEEE Std 2003.1-1992 {4}) GA34

UNUSED

M_GA_EnvironCaseSensitive(function) =

TEST: The interface function retains the unique identities of uppercase and lowercase letters in the environment and does not fold them together.

GA_EnvironCaseSensitive

FOR: *execl(), execle(), execv(), execve(), execlp(), execvp()* and *getenv().*
M_GA_EnvironCaseSensitive*(function())*
Conformance for Environment Description: PASS

(IEEE Std 2003.1-1992 {4}) GA35

UNUSED

M_GA_EnvironPortNames() =

TEST: The interface supports environment variable names consisting of characters in the portable filename character set.

GA_EnvironPortNames

FOR: *execl(), execle(), execv(), execve(), execlp(), execvp()* and *getenv().*
M_GA_EnvironPortNames*(function())*
Conformance for Environment Description: PASS

2.7 C Language Definitions

2.7.1 Symbols From the C Standard

(IEEE Std 2003.1-1992 {4}) 04

UNUSED

4 **TEST:** Each of the headers `<aio.h>`, `<dirent.h>`, `<fcntl.h>`, `<grp.h>`, `<limits.h>`, `<locale.h>`, `<mqueue.h>`, `<pwd.h>`, `<sched.h>`, `<semaphore.h>`, `<signal.h>`, `<sys/mman.h>`, `<sys/stat.h>`, `<sys/times.h>`, `<sys/wait.h>`, `<termios.h>`, `<time.h>`, `<unistd.h>`, and `<utime.h>` can be included more than once, in any combination, in any order, and a symbol may be defined in more than one header with the same value.

NOTE: The C Standard {2} headers that do not have additional requirements placed on them by IEEE Std 1003.1b-1993 are not included, because their testing should be done when measuring conformance to the C Standard {2}.

Conformance for C Language Definitions: PASS

4.1 **TEST:** The header `<sys/types.h>` can be included more than once, in any combination with other headers so long as the first instance of its inclusion precedes any other header that depends upon its prior inclusion; also a symbol may be defined in more than one header with the same value.

Conformance for C Language Definitions: PASS

2.7.2 POSIX.1 Symbols

(IEEE Std 2003.1-1992 {4}) 05

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

UNUSED

5 **FOR:** Headers <aio.h>, <dirent.h>, <fcntl.h>, <grp.h>, <limits.h>, <locale.h>, <mqueue.h>, <pwd.h>, <sched.h>, <semaphore.h>, <signal.h>, <sys/mman.h>, <sys/stat.h>, <sys/times.h>, <>sys/wait.h>, <termios.h>, <time.h>, <unistd.h> and <utime.h>

IF the feature test macro _POSIX_C_SOURCE is defined to have at least the value 199309L **THEN**

TEST: All symbols required by IEEE Std 1003.1b_1993 to appear when a header is included shall be made visible when the _POSIX_C_SOURCE feature test macro is defined.

NOTE: The assertion test would require an unreasonable amount of time or resources on most implementations.

ELSE *NO_OPTION*

Conformance for C Language Definitions: PASS, NO_TEST, NO_OPTION

6 **FOR:** <aio.h>, <dirent.h>, <fcntl.h>, <grp.h>, <limits.h>, <locale.h>, <mqueue.h>, <pwd.h>, <sched.h>, <semaphore.h>, <signal.h>, <sys/mman.h>, <sys/stat.h>, <sys/times.h>, <>sys/wait.h>, <termios.h>, <time.h>, <unistd.h> and <utime.h>

TEST: When a header is included, additional symbols not required or explicitly permitted by IEEE Std 1003.1b-1993 or the C Standard {2} to be in that header shall not be made visible, except when enabled by another feature test macro or by having defined _POSIX_C_SOURCE with a value larger than 199309L

NOTE: The assertion test would require an unreasonable amount of time or resources on most implementations.

Conformance for C Language Definitions: PASS, NO_TEST

(IEEE Std 2003.1-1992 {4})C01

UNUSED

D_1 **IF** the IUT supports feature test macros in addition to _POSIX_C_SOURCE **THEN**

TEST: The PCD.1b either documents the additional feature test macros in clause 2.7.2 or it does not document them at all.

ELSE *NO_OPTION*

Conformance for C Language Definitions: PASS, NO_OPTION

2.7.2.1 C Standard Language-Dependent Support

7 **SETUP:** A program does not use any feature test macros.

TEST: The IUT makes visible only those identifiers specified as reserved identifiers in the C Standard {2}.

NOTE: The assertion test requires setup procedures that involve an unreasonable amount of effort by the user of a test method.

Conformance for C Language Definitions: PASS, NO_TEST

8 **FOR:** Each feature test macro present.

TEST: The IUT makes visible only those identifiers specified by that feature test macro and those of the C Standard {2} when a header is included.

NOTE: The assertion test requires setup procedures that involve an unreasonable amount of effort by the user of a test method.

Conformance for C Language Definitions: PASS, NO_TEST

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.7.2.2 Common-Usage-Dependent Support

9 **SETUP:** A program defines _POSIX_C_SOURCE before any header is included.
TEST: No symbols other than those from the C Standard {2} and those made visible by feature test macros defined for the program (including _POSIX_C_SOURCE) are visible, except that symbols from the namespace reserved for the implementation, as defined by the C Standard {2}, are also permitted.
NOTE: The symbols beginning with two underscores are examples of this.

The assertion test requires setup procedures that involve an unreasonable amount of effort by the user of a test method.
Conformance for C Language Definitions: PASS, NO_TEST

2.7.3 Headers and Function Prototypes

(IEEE Std 2003.1-1992 {4} GA36
UNUSED

M_GA_stdC_proto_decl(func_type; function; parameters; header1; header2; header3; header4) =
IF standard **THEN**
SETUP: The headers `<header1>, <header2>, <header3>,` and `<header 4>` are included.
TEST: The function prototype *func_type function (parameters)* is declared.
ELSE *NO_OPTION*

GA_stdC_proto_decl
FOR: All elements except *assert* (), *setjmp* (), and *sigsetjmp* ().
M_GA_stdC_proto_decl(func_type; function; parameters; header1; header2; header3; header 4)
Conformance for C Language Definitions: PASS, NO_OPTION

M_GA_commonC_result_decl(func_type; function; header1; header2; header3; header4) =
IF the implementation does not provide C Standard {2} support **THEN**
SETUP: The headers `<header1>, <header2>, <header3>,` and `<header 4>` are included.
TEST: The function *function* () is declared with the result type *func_type,* or an equivalent type if the result type is *void.*
ELSE *NO_OPTION*

GA_commonC_result_decl
FOR: All elements with a result type other than *int.*
M_GA_commonC_result_decl(func_type; function; header1; header2; header3; header4)
Conformance for C Language Definitions: PASS, NO_OPTION

M_GA_commonC_int_result_decl(func-type; unction; header1; header2; header3; header4) =
IF the implementation does not provide C Standard {2} support **THEN**
SETUP: The headers `<function>`, `<header1>`, `<header2>`, and `<header3>` are included.
TEST: The interface *func_type* () is either declared with a result type equivalent to *int* or not declared at all.
ELSE *NO_OPTION*

GA_commonC_int_result_decl
FOR: All elements with a result type of *int.*
M_GA_commonC_int_result-decl(func_type; function; header1; header2; header3; header4)
Conformance for C Language Definitions: PASS, NO_OPTION

M_GA_setjmpDecl( )=
IF the interface *setjmp*() is not defined as a macro **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: The function prototype *int setjmp(jmp_buf env)* is declared with external linkage when the header `<setjmp.h>` is included.

ELSE *NO_OPTION*

GA_setjmpDecl

FOR: *setjmp* ().

M_GA_*setjmpDecl()*

Conformance for C Language Definitions: PASS, NO_OPTION

***M_GA_sigsetjmpDecl()*=**

IF the interface *sigsetjmp*() is not defined as a macro **THEN**

TEST: The function prototype *int sigsetjmp(sigjmp_buf env, int savemask)* is declared with external linkage when the header `<setjmp.h>` is included.

ELSE *NO_OPTION*

GA_sigsetjmpDecl

FOR: *sigsetjmp*().

M_GA_sigsetjmpDecl()

Conformance for C Language Definitions: PASS, NO_OPTION

***M_GA_macro_args(function; header1; header2; header 3: header4)*=**

IF the interface *function*() is defined as a macro **THEN**

SETUP: The headers `<header1>`, `<header2>`, `<header3>`, and `<header4>` are included.

TEST: When the macro *function*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), the macro evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary.

ELSE *NO_OPTION*

GA_macro_args

M_GA_macro_args(function; header1; header2; header3; header4)

Conformance for C Language Definitions: PASS, NO_OPTION

D_2 **IF** the implementation does not provide C Standard {2} support **THEN**

TEST: The PCD.1b documents in clause 2.7.3 the equivalent constructs used when *void* is specified in IEEE Std 1003.1b-1993 as a result type for a function prototype, or it is not documented anywhere.

ELSE *NO_OPTION*

Conformance for C Language Definitions: PASS, NO_OPTION

2.8 Numerical Limits

There are no requirements for conforming implementations in this clause.

2.8.1 C Language Limits

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.8.2 Minimum Values

(IEEE Std 2003.1-1992 {4}) 02

UNUSED

2 **TEST:** The symbols in Table 2-1 shall be defined with the values shown when the header `<limits.h>` is included.

NOTE: Table 2-1 is the same as Table 2-3 in IEEE Std 1003.1b-1993.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for Numerical Limits: PASS

2.8.3 Run-Time Increasable Values

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.8.4 Run-Time Invariant Values (Possibly Indeterminate)

(IEEE Std 2003.1-1992 {4}) 04
UNUSED

(IEEE Std 2003.1-1992 {4}) 05
UNUSED

(IEEE Std 2003.1-1992 {4}) 06
UNUSED

(IEEE Std 2003.1-1992 {4}) 07
UNUSED

(IEEE Std 2003.1-1992 {4}) 08
UNUSED

Table 2-1 – Minimum Values

Name	Description	Value
{_POSIX_AIO_LISTIO_MAX}	The number of I/O operations that can be specified in a list I/O call.	2
{_POSIX_AIO_MAX}	The number of outstanding asynchronous I/O operations.	1
{_POSIX_ARG_MAX}	The length of the arguments for one of the *exec* functions, in bytes, including environment data.	4096
{_POSIX_CHILD_MAX}	The number of simultaneous processes per real user ID.	6
{_POSIX_DELAYTIMER_MAX}	The number of timer expiration overruns.	32
{_POSIX_LINK_MAX}	The value of a file's link count.	8
{_POSIX_MAX_CANON}	The number of bytes in a terminal canonical input queue.	255
{_POSIX_MAX_INPUT}	The number of bytes for which space will be available in a terminal input queue.	255
{_POSIX_MQ_OPEN_MAX}	The number of message queues that can be open for a single process.	8
{_POSIX_MQ_PRIO_MAX}	The maximum number of message priorities supported by the implementation.	32
{_POSIX_NAME_MAX}	The number of bytes in a filename.	14
{_POSIX_NGROUPS_MAX}	The number of simultaneous supplementary group IDs per process.	0
{_POSIX_OPEN_MAX}	The number of files that one process can have open at one time.	16
{_POSIX_PATH_MAX}	The number of bytes in a pathname.	255

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Name	Description	Value
{_POSIX_PIPE_BUF}	The number of bytes that can be written atomically when writing to a pipe.	512
{_POSIX_RTSIG_MAX}	The number of realtime signal numbers reserved for application use.	8
{_POSIX_SEM_NSEMS_MAX}	The number of semaphores that a process may have.	256
{_POSIX_SEM_VALUE_MAX}	The maximum value a semaphore may have.	32767
{_POSIX_SIGQUEUE_MAX}	The number of queued signals that a process may send and have pending at the receiver(s) at any time.	32
{_POSIX_SSIZE_MAX}	The value that can be stored in an object of type *ssize_t*.	32767
{_POSIX_STREAM_MAX}	The number of streams that one process can have open at one time.	8
{_POSIX_TIMER_MAX}	The per-process number of timers.	32
{_POSIX_TZNAME_MAX}	The maximum number of bytes supported for the name of a time zone (not of the **TZ** variable).	3

(IEEE Std 2003.1-1992 {4} 09
UNUSED

(IEEE Std 2003.1-1992 {4} D02
UNUSED

4 **TEST:** The values defined in Table 2-2 are equal to or less than those either defined in `<limits.h>` or provided by the *sysconf*() interface.
NOTE: Table 2-2 is the same as Table 2-5 in IEEE Std 1003.1b-1993.
Conformance for Numerical Limits: PASS

Table 2-2 – Run-Time Invariant Values (Possibly Indeterminate)

Name	Description	Minimum Value
{AIO_LISTIO_MAX}	Maximum number of I/O operations in a single list I/O call supported by the implementation.	{POSIX_AIO_LISTIO_MAX}
{AIO_MAX}	Maximum number of outstanding asynchronous I/O operations supported by the implementation.	{_POSIX_AIO_MAX}
{AIO_PRIO_DELTA_MAX}	The maximum amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority.	0
{ARG_MAX}	Maximum length of arguments for the *exec* functions, in bytes, including environment data.	{_POSIX_ARG_MAX}
{CHILD_MAX}	Maximum number of simultaneous processes per real user ID.	{_POSIX_CHILD_MAX}

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	Description	Minimum Value
{DELAYTIMER_MAX}	Maximum number of timer expiration overruns.	{_POSIX_DELAYTIMER_MAX}
{MQ_OPEN_MAX}	The maximum number of open message queue descriptors a process may hold.	{_POSIX_MQ_OPEN_MAX}
{MQ_PRIO_MAX}	The maximum number of message priorities supported by the implementation.	{POSIX_MQ_PRIO_MAX}
{OPEN_MAX}	Maximum number of files that one process can have open at any given time.	{_POSIX_OPEN_MAX}
{PAGESIZE}	Granularity in bytes of memory mapping and process memory locking.	1
{RTSIG_MAX}	Maximum number of realtime signals reserved for application use in this implementation.	{_POSIX_RTSIG_MAX}
{SEM_NSEMS_MAX}	Maximum number of semaphores that a process may have.	{_POSIX_SEM_NSEMS_MAX}
{SEM_VALUE_MAX}	The maximum value a semaphore may have.	{_POSIX_SEM_VALUE_MAX}
{SIGQUEUE_MAX}	Maximum number of queued signals that a process may send and have pending at the receiver(s) at any time.	{POSIX_SIGQUEUE_MAX}
{STREAM_MAX}	The number of streams that one process can have open at one time. If defined, it shall have the same value as {FOPEN_MAX} from the C Standard {2}.	{_POSIX_STREAM_MAX}
{TIMER_MAX}	Maximum number of timers per process supported by the implementation.	{_POSIX_TIMER_MAX}
{TZNAME_MAX}	The maximum number of bytes supported for the name of a time zone (not of the **TZ** variable).	{_POSIX_TZNAME_MAX}

5 **IF** a definition of one of the values in Table 2-2 is omitted from `<limits.h>`
THEN
TEST: The corresponding omitted value is equal to or greater than the stated minimum in Table 2-2, and the actual value is provided by the *sysconf*() interface.
ELSE *NO_OPTION*
Conformance for Numerical Limits: PASS, NO_OPTION

D_2 TEST: The run-time invariant values for the identifiers specified in Table 2-2 are documented in clause 2.8.4 of the PCD.1b.
Conformance for Numerical Limits: PASS

2.8.5 Pathname Variable Values

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.8.6 Invariant Values

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.8.7 Maximum Values

18 **TEST:** The symbols in Table 2-3 shall be defined in `<limits.h>`, when it is included, and have the values shown.
NOTE: Table 2-3 is the same as Table 2-7a in IEEE Std 1003.1b-1993.
Conformance for Numerical Limits: PASS

Table 2-3 – Maximum Values

Name	Description	Value
{_POSIX_CLOCKRES_MIN}	The CLOCK_REALTIME clock resolution, in nanoseconds.	20 000 000

2.9 Symbolic Constants

(IEEE Std 2003.1-1992 {4} D01
UNUSED

(IEEE Std 2003.1-1992 {4} D02
UNUSED

(IEEE Std 2003.1-1992 {4} D03
UNUSED

(IEEE Std 2003.1-1992 {4} D04
UNUSED

D_1 FOR: Any of the symbols specified in Table 2-4 that are defined in `<unistd.h>`.
TEST: The value associated with the symbol, the conditions under which the value may change, and the limits of such variations are documented in clause 2.9 of the PCD.1b.
NOTE: Table 2-4 is the same as Table 2-10 in IEEE Std 1003.1b-1993.
Conformance for Symbolic Constants: PASS

D-2 **FOR:** Any of the symbols specified in Table 2-5 that are defined in `<unistd.h>`.
TEST: The value associated with the symbol, the conditions under which the value may change, and the limits of such variations are documented in clause 2.9 of the PCD.1b.
NOTE: Table 2-5 is the same is Table 2-11 in IEEE Std 1003.1b-1993.
Conformance for Symbolic Constants: PASS

2.9.1 Symbolic Constants for the *access*() Function

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

2.9.2 Symbolic Constant for the *lseek*() Function

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; no POSIX.1b {3} assertions.

2.9.3 Compile-Time Symbolic Constants for Portability Specifications

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(IEEE Std2003.1-1992 {4} R02
UNUSED

(IEEE Std 2003.1-1992 {4} R03
UNUSED

1 **TEST:** The values of the constants specified in Table 2-4 are not less restrictive than those provided by the corresponding value returned by *sysconf*().
Conformance for Symbolic Constants: PASS

Table 2-4 – Compile-Time Symbolic Constants

Name	Description
{_POSIX_ASYNCHRONOUS_IO}	If this symbol is defined, the implementation supports the Asynchronous Input and Output option.
{_POSIX_FSYNC}	If this symbol is defined, the implementation supports the File Synchronization option.
{_POSIX_JOB_CONTROL}	If this symbol is defined, the implementation supports the Job Control option.
{_POSIX_MAPPED_FILES}	If this symbol is defined, the implementation supports the Memory Mapped Files option.
{_POSIX_MEMLOCK}	If this symbol is defined, the implementation supports the Process Memory Locking option.
{_POSIX_MEMLOCK_RANGE}	If this symbol is defined, the implementation supports the Range Memory Locking option.
{_POSIX_MEMORY_PROTECTION}	If this symbol is defined, the implementation supports the Message Protection option.
{_POSIX_MESSAGE_PASSING}	If this symbol is defined, the implementation supports the Message Passing option.
{_POSIX_PRIORITIZED_IO}	If this symbol is defined, the implementation supports the Prioritized Input and Output option.
{_POSIX_PRIORITY_SCHEDULING}	If this symbol is defined, the implementation supports the Process Scheduling option.
{_POSIX_REALTIME_SIGNALS}	If this symbol is defined, the implementation supports the Realtime Signals Extension option.
{_POSIX_SAVED_IDS}	If this symbol is defined, each process has a saved set-user-ID and a saved set-group-ID.
{_POSIX_SEMAPHORES}	If this symbol is defined, the implementation supports the Semaphores option.
{_POSIX_SHARED_MEMORY_OBJECTS	If this symbol is defined, the implementation supports the Shared Memory Objects option.
{_POSIX_SYNCHRONIZED_IO}	If this symbol is defined, the implementation supports the Synchronized Input and Output option.
{_POSIX_TIMERS}	If this symbol is defined, the implementation supports the Timers option.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

{_POSIX_VERSION}	The integer value 199309L; this value shall be used for systems that conform to this standard.

(IEEE Std 2003.1-1992 {4} 05
UNUSED

2 **TEST:** The symbol {POSIX_VERSION} is defined and has the value 199309L when <unistd.h> is included.
Conformance for Symbolic Constants: PASS

3 **IF** the symbol {_POSIX_MEMLOCK_RANGE} is defined in <unistd.h> **THEN**
TEST: The symbol {_POSIX_MEMLOCK} shall be defined in <unistd.h>
ELSE *NO_OPTION*
Conformance for Symbolic Constants: PASS, NO-OPTION

4 **IF** the symbol {_POSIX_MEMORY_PROTECTION} is defined in <unistd.h>
THEN
TEST: At least one of the symbols {_POSIX_MAPPED_FILES} or {_POSIX_SHARED_MEMORY_OBJECTS} shall be defined in <unistd.h>.
ELSE *NO_OPTION*
Conformance for Symbolic Constants: PASS, NO_OPTION

5 **IF** the symbol {_POSIX_SYNCHRONIZED_IO} is defined in <unistd.h> **THEN**
TEST: The symbol {_POSIX_FSYNC} shall be defined in <unistd.h>.
ELSE *NO_OPTION*
Conformance for Symbolic Constants: PASS, NO_OPTION

2.9.4 Execution-Time Symbolic Constants for Portability Specifications

(IEEE Std 2003.1-1992 {4}) D05
UNUSED

(IEEE Std 2003.1-1992 {4}) D06
UNUSED

(IEEE Std 2003.1-1992 {4}) D07
UNUSED

(IEEE Std 2003.1-1992 {4}) R04
UNUSED

(IEEE Std 2003.1-1992 {4}) R05
UNUSED

(IEEE Std 2003.1-1992 {4}) R06
UNUSED

(IEEE Std 2003.1-1992 {4}) 06
UNUSED

(IEEE Std 2003.1-1992 {4}) 07
UNUSED

(IEEE Std 2003.1-1992 {4}) 08
UNUSED

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6 **TEST:** The PCD.1b documents in clause 2.9.4 whether each of the values associated with the symbols in Table 2-5 are defined in the header `<unistd.h>`, and if each value defined is -1 or other than -1.
NOTE: Table 2-5 is the same as Table 2-11 in IEEE Std 1003.1b-1993.
Conformance for Symbolic Constants: PASS

7 **FOR:** Any of the symbols in Table 2-5 that have the value -1 in the header `<unistd.h>`.
TEST: The IUT shall not provide the corresponding option on any file.
NOTE: Table 2-5 is the same as Table 2-11 in IEEE Std 1003.1b-1993.

There is no known reliable test method for this assertion.
Conformance for Symbolic Constants: PASS, NO_TEST

8 **FOR:** Any of the symbols in Table 2-5 that have a value other than -1 in the header `<unistd.h>`.
TEST: The IUT shall provide the corresponding option on all applicable files.
NOTE: Table 2-5 is the same as Table 2-11 in IEEE Std 1003.1b-1993.

There is no known reliable test method for this assertion.
Conformance for Symbolic Constants: PASS, NO_TEST

Table 2-5 – Execution-Time Symbolic Constants

Name	Description
{_POSIX_ASYNC_IO}	Asynchronous input or output operations may be performed for the associated file.
{_POSIX_SHOWN_RESTRICTED}	The implementation supports the Change File Owner Restriction. The use of the *chown*() function is restricted to a process with appropriate privileges, and to changing the group ID of a file only to the effective group ID of the process or to one of its supplementary group IDs.
{_POSIX_NO_TRUNC}	Pathname components longer than {NAME_MAX} generate an error.
{_POSIX_PRIO_IO}	Prioritized input or output operations may be performed for the associated file.
{_POSIX_SYNC_IO}	Synchronized input or output operations may be performed for the associated file.
{_POSIX_VDISABLE}	Terminal special characters defined in 7.1.1.9 in POSIX.1b {3} can be disabled using this character value, if it is defined. See *tcgetattr*() and *tcsetattr*().

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Section 3: Process Primitives

3.1 Process Creation and Execution

3.1.1 Process Creation

Function: *fork*().

3.1.1.1 Symbols

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.1.1.2 Description

1 **IF** *PCTS_sem_init*() **THEN**
TEST: Any semaphores that are open in the parent process when it makes a *fork*() call shall also be open in the child process.
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_OPTION

2 **IF** *PCTS_sem_open* **THEN**
IF *PCTS_GAP_sem_init* **THEN**
TEST: Any semaphores that are open in the parent process when it makes a *fork*() call shall also be open in the child process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_TEST_SUPPORT, NO_OPTION

3 **FOR:** *mlock*() and *mlockall*()
IF *PCTS_function* **THEN**
TEST: A child process shall not inherit any address space memory locks established by the parent process via calls to *function*() after a *fork*() call.
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function* () with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function*.
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_OPTION

4 **IF** *PCTS_mmap* **THEN**
TEST: Memory mappings created in the parent process are retained in the child process after a *fork*() call.
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5 **IF** *PCTS_mmap* **THEN**
TEST: MAP_PRIVATE mappings inherited from the parent after a *fork*() call shall also be MAP_PRIVATE mappings in the child, and any modifications to the data in these mappings made by the parent prior to calling *fork*() shall be visible to the child.
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_OPTION

6 **IF** *PCTS_mmap* **THEN**
TEST: Any modifications to the data in MAP_PRIVATE mappings made by the parent after *fork*() returns shall be visible only to the parent.
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_OPTION

7 **IF** *PCTS_mmap* **THEN**
TEST: Modifications to the data in MAP_PRIVATE mappings made by the child shall be visible only to the child.
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_OPTION

8 **IF** *PCTS_sched_setschedulerI or PCTS_sched_setparam* **THEN**
IF *PCTS_sched_getscheduler* or *PCTS_sched_getparam* **THEN**
TEST: For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit the policy and priority settings of the parent process during a *fork*() function.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_sched_setscheduler* or *PCTS_sched_setparam* **THEN**
TEST: The PCD.1b documents the policy and priority settings on *fork*() for all scheduling policies other than SCHED_FIFO and SCHED_RR in 3.1.1.2.
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_OPTION

10 **IF** *PCTS_timer_create* **THEN**
IF *PCTS_timer_settime* and *PCTS_timer_gettime* **THEN**
TEST: Per-process timers created by the parent are not inherited by the child process after a *fork*() call.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_mq_send* and *PCTS_mq_receive* **THEN**
TEST: A child process has its own copy of the message queue descriptors of its parent, and each of the message queue descriptors of the child refers to the same open message queue description as the corresponding message descriptor of the parent.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_TEST_SUPPORT, NO_OPTION

12 **IF** {_POSIX_ASYNCHRONOUS_IO} **THEN**
TEST: No asynchronous input or asynchronous output operations are inherited by the child process after a *fork*() call.
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_TEST, NO_OPTION

13 **FOR:** *PCTS_aio_read, PCTS_aio_write, PCTS_lio_listio*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

IF *PCTS_function* **THEN**
IF *PCTS_aio_cancel,* **THEN**
TEST: Asynchronous input or asynchronous output operations created by calling *function* () are not inherited by the child process after a *fork* () call.
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fork: PASS, NO_TEST_SUPPORT, NO_OPTION

3.1.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.1.1.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.1.2 Execute a File

Functions: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*().

3.1.2.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.1.2.1 Description

1 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*().
IF *PCTS_sem_open* **THEN**
SETUP: Open a named semaphore, then call *function*().
TEST: Any named semaphores that are open in the calling process shall be closed as if by appropriate calls to *sem_close*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_OPTION

2 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*().
IF *(PCTS_mlockall* and *PCTS_GAP_mlockall*) or (*PCTS_mlock* and *PCTS_GAP_mlock*) **THEN**
SETUP: Establish memory locks before calling *function*().
TEST: Memory locks are removed after a call to *function*().
TR: Establish the memory locks using as many of the interfaces *mlockall*() and *mlock*() as are implemented.
NOTE: The interface *munlock*() can be used in the program loaded by *function()* to determine whether or not memory locks were removed.
ELSE NO_OPTION
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_OPTION

3 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF *PCTS_mlockall* and *PCTS_mlock* **THEN**
IF *PCTS_GAP_mlockall* and *PCTS_GAP_mlock* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SETUP: Create locked pages in the address space of the process that will call *function*(), and also map and lock the same pages into the address space of another process.

TEST: The memory page locks for memory pages that are mapped into the address space of other processes and locked by them are unaffected by a process that has locks on those same pages and that calls *function* ().

TR: Establish the memory locks using as many of the interfaces *mlockall*() and *mlock*() as are implemented.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 FOR: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()

IF *PCTS_mmap* or *PCTS_shm_open* and a PCD.1b documents the following **THEN**

TEST: PCD.1b that documents the effect on memory locks in the *function*() fails does so in 3.1.2.2.

ELSE *NO_OPTION*

Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_OPTION

4 FOR: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()

IF *PCTS_mmap* or *PCTS_shm_open* **THEN**

TEST: Memory mappings created in a process are unmapped before the address space is rebuilt for the new process image after a call to *function*().

NOTE: There is no known portable test method for this assertion.

ELSE *NO_OPTION*

Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_TEST, NO_OPTION

5 FOR: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()

IF *PCTS_sched_setscheduler* and *PCTS_getscheduler* **THEN**

TEST: The policy and priority settings for the SCHED_FIFO and SCHED_RR scheduling policies are not changed for a process that calls *function*().

ELSE *NO_OPTION*

Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_OPTION

D_2 FOR: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()

IF *PCTS_sched_setscheduler* and *PCTS_getscheduler* **THEN**

TEST: The PCD.1b documents for scheduling policies, other than SCHED_FIFO and SCHED_RR, the policy and priority settings after a call to *function*() in 3.1.2.2.

ELSE *NO_OPTION*

Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_OPTION

6 FOR: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()

IF *PCTS_timer_create and PCTS_timer_settime* **THEN**

TEST: Per-process timers created by the calling process are deleted before replacing the current process image with the new process image after a call to *function*().

NOTE: There is no known portable test method for this assertion.

ELSE *NO_OPTION*

Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_TEST, NO_OPTION

7 FOR: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()

IF *PCTS_timer_create and PCTS_timer_settime* **THEN**

TEST: Per-process timers created by the calling process are deleted after a call to *function*().

NOTE: This can be tested by setting a timer before the *function*() call and waiting in the new process image for the timer's signal. If no signal arrives after a sufficiently long time, the timer was destroyed or the equivalent. To determine of the timer was actually destroyed, try to create {TIMER_MAX} timers. If able to do so, the timer has been destroyed.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_OPTION

8 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF *PCTS_mq_open* **THEN**
IF *PCTS_mq_close* and *PCTS_mq_send* and *PCTS_mq_receive* **THEN**
SETUP: Create two processes; one will call *function*(), and the other will test the message queue with the new process image created by *function*().
TEST: After a call to *function*(), all open message queue descriptors in the calling process shall be closed, and the association between the message queue descriptor and the message queue is removed.
TR: Test for at least two message queues.
NOTE: This may not be testable if the use of message queue descriptors is undefined after a close.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF *PCTS_mq_open* **THEN**
IF ({_POSIX_MESSAGE_PASSING} and {_POSIX_REALTIME_SIGNALS}) or (*PCTS_mq_close* and *PCTS_mq_send* and *PCTS_mq_notify*) **THEN**
SETUP: Create two processes; one will call *function*(), and the other will test the message queue with the new process image created by *function*().
TEST: After a call to *function*(), all open message queue descriptors in the calling process shall be closed; an attached message queue notification request is removed, and the message queue is available for another process to attach a notification.
NOTE: This may not be testable if the use of message queue descriptors is undefined after a close.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_TEST_SUPPORT, NO_OPTION

10 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF {_POSIX_ASYNCHRONOUS_IO} **THEN**
TEST: Any asynchronous I/O operations that are not canceled after calling *function*() complete as if the *function*() call had not yet occurred, but any associated signal notifications are suppressed.
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_TEST, NO_OPTION

D_3 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF {_POSIX_ASYNCHRONOUS_IO} and PCD.1b documents the following **THEN**
TEST: PCD.1b that documents whether the *function*() itself blocks awaiting asynchronous I/O completion does so in 3.1.2.2.
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_OPTION

11 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF {_POSIX_ASYNCHRONOUS_IO} **THEN**
TEST: The new process image created after a *function*() call is not affected by the presence of outstanding asynchronous I/O operations at the time the *function*() is called.
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_TEST, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D_4 FOR: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF {_POSIX_ASYNCHRONOUS_IO} **THEN**
TEST: The PCD.1b documents whether any asynchronous I/O operation is canceled, and which I/O may be canceled upon a call to *function*(), in 3.1.2.2.
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_OPTION

3.1.2.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b assertions.

3.1.2.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.2 Process Termination

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.2.1 Wait for Process Termination

Functions: *wait*(), *waitpid*().

3.2.1.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.2.1.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.2.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.2.1.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.2.2 Terminate a Process

Function: *_exit*().

3.2.2.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.2.2.2 Description

1 **IF** *PCTS_sem_open* **THEN**
IF *PCTS_sem_trywait* and *PCTS_sem_getvalue* **THEN**
SETUP: Create {PCTS_SEM_NSEMS_MAX} named semaphores with large initial values and lock each one at least once.
TEST: After a call to *_exit*(), all open named semaphores in the calling process have no effect on the state of such semaphores.
ELSE *NO_TEST_SUPPORT*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
Conformance for _exit: PASS, NO_TEST_SUPPORT, NO_OPTION

2 **FOR:** *mlock*() and *mlockall*()
IF *PCTS_function*() **THEN**
IF *PCTS_GAP_function*() **THEN**
SETUP: Lock all pages of the process in memory.
TEST: Any memory locks established by the process via calls to *function*() are removed after a call to *_exec*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for _exit: PASS, NO_TEST_SUPPORT, NO_OPTION

3 **FOR:** *mlock*() and *mlockall*()
IF *PCTS_function*() **THEN**
IF *PCTS_GAP_function*() **THEN**
SETUP: Use *function*() to create locked pages in the address space of the process calling *_exit*() that are also mapped into the address spaces of other processes and are locked by them.
TEST: The memory locks established by the other processes are unaffected by the call by this process to -*exit*().
TR: Test for at least two other processes.
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function*.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for _exit: PASS, NO_TEST_SUPPORT, NO_OPTION

4 **IF** *PCTS_mmap* **THEN**
TEST: Memory mappings created in the process are unmapped before the process is destroyed after a call to *_exec*().
ELSE *NO_OPTION*
Conformance for _exit: PASS, NO_TEST, NO_OPTION

5 **IF** *PCTS_mq_open* **THEN**
IF *PLCTS_mq_notify* **THEN**
SETUP: Create a message queue, a process to call *exit*(), and another process to check on the state of the message queue. Call *mq_notify*() from the process that will call *_exit*().
TEST: All open message queue descriptors in the process calling *_exit*() are closed, which allows other processes to issue successful *mq_notify*() calls.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for _exit: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** {_POSIX_ASYNCHRONOUS_IO} **THEN**
TEST: Those asynchronous I/O operations that are not canceled after a call to *_exit*() completes, as if the *_exit*() operation had not yet occurred, but any associated signal notifications are suppressed.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for _exit: PASS, NO_TEST, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D_1 IF a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not the *_exit*() operation itself blocks awaiting the completion of asynchronous I/O does so in 3.2.2.2.

ELSE *NO_OPTION*

Conformance for _exit: PASS, NO_OPTION

D_2 TEST: The PCD.1b documents whether any asynchronous I/O is canceled, and which asynchronous I/O may be canceled upon a call to *_exit*() in 3.2.2.

Conformance for _exit: PASS

3.2.2.3 Returns

There are no requirements for conforming implementations in this clause.

3.3 Signals

3.3.1 Signal Concepts

3.3.1.1 Signal Names

NOTE: Tables 3-1, 3-2, and 3-3 are kept in the same order in this clause as in POSIX.1b {3} to facilitate the following the correspondence between assertions in this standard and the requirements specified in POSIX.1b {3}.

Table 3-1 – Required Signals

Symbolic Constant	Default Action	Description
SIGABRT	1	Abnormal termination signal, such as is initiated by the *abort*() function.
SIGALRM	1	Timeout signal, such as initiated by the *alarm*() function.
SIGFPE	1	Erroneous arithmetic operations, such as division by zero or an operation resulting in overflow.
SIGHUP	1	Hangup detected on controlling terminal or death of controlling process.
SIGINT	1	Interactive attention signal.
SIGKILL	1	Termination signal (cannot be caught or ignored).
SIGPIPE	1	Write on a pipe with no readers.
SIGQUIT	1	Interactive termination signal.
SIGSEGV	1	Detection of an invalid memory reference.
SIGTERM	1	Termination signal.
SIGUSR1	1	Reserved as application-defined signal 1.
SIGUSR2	1	Reserved as application-defined signal 2.

NOTE: The default actions are

1 Abnormal termination of the process.

1 **SETUP:** Include the header `<signal.h>`.

TEST: The constants shown in Table 3-3 are defined.

Conformance for signal.h: PASS

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**
SETUP: Include the header <signal.h>.
TEST: The signals shown in Table 3-3 behave with the specified default action.

Table 3-2 – Job Control Signals

Symbolic Constant	Default Action	Description
SIGCHLD	2	Child process terminated or stopped.
SIGCONT	4	Continue if stopped.
SIGSTOP	3	Stop signal (cannot be caught or ignored).
SIGTSTP	3	Interactive stop signal.
SIGTTIN	3	Read from control terminal attempted by a member of a background process group.
SIGTTOU	3	Write to control terminal attempted by a member of a background process group.

NOTE: The default actions are

2 Ignore the signal.

3 Stop the process.

4 Continue the process if it is currently stopped; otherwise, ignore the signal.

Table 3-3 – Memory Protection Signals

Symbolic Constant	Default Action	Description
SIGBUS	1	Access to an undefined portion of a memory object.

TR: Test for child processes and processes that are not children of the calling process.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

3 **SETUP:** Include the header <signal.h>.
TEST: The macros SIGRTMIN and SIGRTMAX are defined and evaluate to integral expressions.
Conformance for signal.h: PASS

4 **SETUP:** Include the header <signal.h>.
TEST: The signal numbers in the range SIGRTMIN to SIGRTMAX do not overlap with any of the signals specified in Tables 3-1, 3-2, or 3-3.
Conformance for signal.h: PASS

5 **SETUP:** Include the header <signal.h>.
TEST: The range SIGRTMIN through SIGRTMAX inclusive includes at least {RTSIG_MAX} signal numbers.
Conformance for signal.h: PASS

D_1 TEST: The PCD.1b documents whether the realtime signal behavior for the queuing of signals and the passing of application defined values is supported for each of the signals defined in Tables 3-1, 3-2, and 3-3 in 3.3.1.1.
Conformance for signal.h: PASS

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.3.1.2 Signal Generation and Delivery

6 **SETUP:** Include the header `<signal.h>`.
TEST: The *sigevent* structure is defined, *sigev_value* is equal to or greater than SIGRTMIN or less than SIGRTMAX, and contains at least the following members:

Member Type	Member Name	Description
int	*sigev_notify*	Notification type
int	*sigev_signo*	Signal number
union sigval	*sigev_value*	Signal value

Conformance for signal.h: PASS

D_2 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents any extensions that are added to the *sigevent* structure, and how they are enabled, does so in 3.3.1.2.
TR: Only extensions permitted in 1.3.1.1, item (2) in IEEE Std 1003.1b-1993 may be added.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

7 **SETUP:** Include the header `<signal.h>`.
TEST: The symbols SIGEV_NONE and SIGEV_SIGNAL are defined.
Conformance for signal.h: PASS

D_3 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents additional notification mechanisms to use when asynchronous events occur does so in 3.3.1.2.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

M_GA_sigev_value() =
IF {_POSIX_REALTIME_SIGNALS} **THEN**
TEST: The *sigev_value* member of the *sigevent* structure is passed to a signal-catching function at the time of the signal delivery as the *si_value- member of the siginfo_t* structure.
TR: Test for all catchable signals.
ELSE *NO_OPTION*

GA_sigev_value
FOR: *lio_listio*(), *timer_create*(), *mq_notify*(), *aio_read*(), *aio_write*(), and *aio_fsync*()
M_GA_sigev_value ()
Conformance for signal.h: PASS, NO_OPTION

M_GA_sigqueueValue() =
IF {_POSIX_REALTIME_SIGNALS} **THEN**
TEST: The *value* parameter to *sigqueue*() is passed to a signal-catching function at the time of the signal delivery as the *si_value* member of the *siginfo_t* structure.
TR: Test for all catchable signals.
ELSE *NO_OPTION*

GA_sigqueueValue
FOR: *siqgueue*()
M_GA_sigqueueValue()
Conformance for signal.h: PASS, NO_OPTION

8 **SETUP:** Include the header `<signal.h>`.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: The *sigval* is defined and contains at least the following members:

Member Type	Member Name	Description
int	*sival_int*	Integer signal value
*void**	*sival_ptr*	Pointer signal value

Conformance for signal.h: PASS

***M_GA_sigPending(function)*=**

SETUP: Cause the *function*() interface to generate a signal after having set an application-specified value to be transmitted along with the signal.

TEST: After a signal is generated by the *function*() interface the signal shall be marked pending and, if the SA_SIGINFO flag is set for that signal, the signal shall be queued to the process along with the application-specified signal value.

TR: Test for all catchable signals both with SA_SIGINFO set and not set.

GA_sigPending

FOR: *sigqueue*(), *lio_listio*(), *timer_create*(), *mq_notify*(), *aio_read*(), *aio_write*(), and *aio_fsync*()

M_GA_sigPending(function)

Conformance for signal.h: PASS, NO_OPTION

***M_GA_sigPendingQueued(function)*=**

SETUP: Cause the *function*() interface to generate a series of signals after having set a different application-specified value for each signal to be transmitted along with the signal.

TEST: Multiple occurrences of signals generated by the *function*() interface, with the SA_SIGINFO flag set for that signal, will be queued in FIFO order.

TR: Test for all catchable signals.

GA_sigPendingQueued

FOR: sigqueue(), *lio_listio*(), *timer_create*(), *mq_notify*(), *aio_read*(), *aio_write*(), and *aio_fsync*()

D_4 IF a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether signals generated by the *sigqueue*() function, or any signal-generating function that supports the specification of an application-defined value, are queued when the SA_SIGINFO flag is not set for that signal, and does so in 3.3.1.2.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

***M_GA_queuedAndRegularSignals(function)*=**

SETUP: Cause the *function*() interface to generate a signal after having set a different application-specified value for each signal to be transmitted along with the signal.

TEST: Signals generated by the *kill*() function or other events that cause signals to occur, such as *alarm*() timer expiration or terminal activity, and for which the implementation does not support queuing, have no effect on signals already queued for the same signal number.

TR: Test for all catchable signals and all the conditions specified in the **TEST.**

GA_queuedAndRegularSignals

FOR: *sigqueue*(), *lio_listio*(), *timer_create*(), *mq_notify*(), *aio_read*(), *aio_write*(), and *aio_fsync*()

M_GA_queuedAndRegularSignals(function)

Conformance for signal.h: PASS, NO_OPTION

9 IF *_POSIX_REALTIME_SIGNALS* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IF *PCTS_sigqueue* **THEN**
SETUP: Create multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, that are pending.
TEST: The implementation delivers the pending unblocked signal with the lowest signal number within that range.
TR: Test with each signal in the range of the lowest unblocked signal.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

10 **IF** *_POSIX_REALTIME_SIGNALS* **THEN**
IF *PCTS_sigqueue* **THEN**
SETUP: Create a pending signal and additional signals queued to the same signal number.
TEST: After a pending signal is delivered, the signal shall remain pending until all queued signals have been delivered, at which time the pending indication is reset.
TR: Test for all catchable signals.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *_POSIX_REALTIME_SIGNALS* **THEN**
IF *PCTS_sigqueue* **THEN**
SETUP: Create a pending signal.
TEST: After a pending signal is delivered and there are no queued signals to the same signal number, the pending indication is reset.
TR: Test for all catchable signals.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

3.3.1.3 Signal Actions

12 **IF** {_POSIX_REALTIME_SIGNALS} **THEN**
IF *PCTS_sigqueue* **THEN**
TEST: The default action for the realtime signals in the range SIGRTMIN through SIGRTMAX is to terminate the process abnormally.
TR: Test for all signals in the range.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** {_POSIX_JOB_CONTROL} and {_POSIX_REALTIME_SIGNALS} **THEN**
TEST: Setting a signal action to SIG_DEL for a SIGCHLD signal that is pending causes the pending signal to be discarded, whether or not it is blocked; in addition, any queued values pending are discarded.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

14 **IF** {_POSIX_JOB_CONTROL} and {_POSIX_REALTIME_SIGNALS} **THEN**
TEST: Setting a signal action to SIG_DFL for a SIGCHLD signal that is pending causes the resources used to queue any pending signals and values to be released and made available to queue other signals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_TEST, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D_5 IF A PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the behavior of a process after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that was not generated by the *kill*() function, the *sigqueue*() function, or the *raise*() function as defined by the C Standard {2} does so in 3.3.1.3.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

15 **IF** {_POSIX_REALTIME_SIGNALS} **THEN**

IF *PCTS_sigqueue* **THEN**

TEST: Setting a signal action to SIG_IGN for a signal that is pending causes the pending signal to be discarded, whether or not it is blocked; in addition, any queued values pending are discarded.

TR: Test for all catchable signals consistent with the implemented options.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **IF** {_POSIX_REALTIME_SIGNALS} **THEN**

IF *PCTS_sigqueue* **THEN**

TEST: If the implementation limits the number of outstanding queued signals to {SIGQUEUE_MAX}, this test can verify that the signal can be sent after setting the action to SIG_IGN in the target process. Setting a signal action to SIG_IGN for a signal that is pending causes the resources used to queue the pending signal and its associated value to be released and made available to queue other signals.

NOTE: There is no known portable test method for this assertion.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

17 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**

IF *PCTS_sigqueue* **THEN**

TEST: Upon delivery of a signal specified in Table 3-3, the receiving process executes the signal-catching function at the specified address.

TR: Test for all signals in Table 3-3.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_sigqueue* **THEN**

SETUP: Clear the SA_SIGINFO flag and establish a signal-catching function based on the following prototype:

```
void func(int signo);
```

TEST: The signal-catching function is passed *signo* as a parameter.

TR: Test for all catchable signals consistent with the implemented options.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

19 **IF** *PCTS_sigqueue* **THEN**

SETUP: Set the SA_SIGINFO flag and establish a signal-catching function based on the following prototype:

```
void func(int signo, siginfo_t *info,
        void *context);
```

TEST: The signal-catching function is passed *signo* (the signal number of the signal being delivered) and *info* (a pointer to a *siginfo_t* structure).

TR: Test for all catchable signals consistent with the implemented options.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

20 **SETUP:** Include the header `<signal.h>`.

TEST: A structure type *siginfo_t* is defined and contains at least the following members:

Member Type	Member Name	Description
int	*si_signo*	Signal number
int	*si_code*	Cause of the signal
union sigval	*si_value*	Signal value

Conformance for signal.h: PASS

21 **IF** *PCTS_sigqueue* **THEN**

SETUP: With the SA_SIGINFO flag set, create a signal-catching function; then generate a signal for that signal-catching function to handle.

TEST: The *si_signo* member of the *siginfo_t* structure contains the signal number and the signal number is the same as the *signo* parameter passed to the signal-catching function.

TR: Test for all catchable signals consistent with implemented options.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

22 **SETUP:** With the SA_SIGINFO flag set, create a signal-catching function and set an application-specific value to be passed to it. Then generate a signal for that signal-catching function to handle by calling the *kill*() function.

TEST: The *si_code* member of the *siginfo_t* structure contains SI_USER and the *si_signo* member of the *siginfo_t* structure contains the signal number and the signal number is the same as the *signo* parameter passed to the signal-catching function.

Conformance for signal.h: PASS

D_6 **IF** A PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether the *si_code* number of the *siginfo_t* structure is set to SI_USER, if the signal was sent by the *raise*() or *abort*() functions as defined in the C Standard {2}, or any similar functions provided as implementation extensions, does so in 3.3.1.3.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

23 **IF** *PCTS_SIGQUEUE* **THEN**

SETUP: With the SA_SIGINFO flag set, declare a signal-catching function and choose an application-specific value to be passed to it. Then generate a signal for that signal-catching function to handle by calling the SI_QUEUE function.

TEST: The *si_code* member of the *siginfor_t* structure contains SI_QUEUE, a code identifying the cause of the signal as being generated by the *sigqueue*() function,

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

and the *si_signo* member of the *siginfo_t* structure contains the signal number and the signal number is the same as the *signo* parameter passed to the signal-catching function.

ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

24 **IF** *PCTS_timer_settime* **THEN**
IF *PCTS_timer_create* **THEN**
SETUP: With the SA_SIGINFO flag set, declare a signal-catching function and choose an application-specific value to be passed to it. Then generate a signal for that signal-catching function to handle by setting up a timer expiration for a timer set by *timer_settime*().
TEST: The *se_code* member of the *siginfo_t* structure contains SI_TIMER, a code identifying the cause of the signal generation as the expiration of a timer set by *timer_settime*(), and the *si_signo* member of the *siginfo_t* structure contains the signal number and it is the same as the *signo* parameter passed to the signal-catching function.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

25 **IF** {_POSIX_ASYNCHRONOUS_IO} and {POSIX_REALTIME_SIGNALS) **THEN**
SETUP: With the SA_SIGINFO flag set, declare a signal-catching function and choose an application-specific value to be passed to it. Then generate a signal for that signal-catching function to handle by completing an asynchronous I/O request.
TEST: The *si_code* member of the *siginfo_t* structure contains SI_ASYNCIO, a code identifying the cause of the signal generation as the completion of an asynchronous I/O request, and the *si_signo* member of the *siginfo_t* structure contains the signal number and it is the same as the *signo* parameter passed to the signal-catching function.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

26 **IF** {_POSIX_MESSAGE_PASSING} and {_POSIX_REALTIME_SIGNALS} **THEN**
SETUP: With the SA_SIGINFO flag set, declare a signal-catching function and choose an application-specific value to be passed to it. Then generate a signal for that signal-catching function to handle by making a message arrive on an empty message queue.
TEST: The *si_code* member of the *siginfo_t* structure contains the code SI_MESGQ, which means the signal was generated by the arrival of a message on an empty message queue, and the *si_signo* member of the *siginfo_t* structure contains the signal number and it is the same as the *signo* parameter passed to the signal-catching function.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

D_7 TEST: The PCD.1b documents the implementation-defined value, which is not equal to any of the values SI_USER, SI_QUEUE, SI_TIMER, SI_ASYNCIO, and SI_MESGQ, and which is also set in the *si_code* if a signal was not generated by one of the following functions or events:

1. The *kill*() function
2. The *raise*() or *abort*() functions as defined in the C Standard {2}, if they set *si_code* to SI_USER
3. The *sigqueue*() function
4. The *timer_settime*() function

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5. The completion of an asynchronous I/O request

6. The arrival of a message on an empty message queue in 3.3.1.3

Conformance for signal h: PASS

27 **IF** *PCTS_MORE_SA_SIGINFO_SIGNALS* and {_POSIX_REALTIME_SIGNALS) **THEN**

SETUP: Establish a signal-catching function with the SA_SIGINFO flag set. Then generate that signal by a means other than calling *kill*(), *raise*(), and *abort*() (if they set to *sigqueue*(), or *timer_settime*() or by completion of an asynchronous I/O request or by the arrival of a message on an empty message queue.

TEST: The *si_code* is set to an implementation-defined value that is not equal to any of the values defined in IEEE Std 1003.1b-1993 for *si_code*.

TR: Test for each such implementation-defined value.

NOTE: It is possible to perform this test by calling a target system-specific function that contains the tests.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

28 **IF** {_POSIX_REALTIME_SIGNALS} **THEN**

IF *PCTS_sigqueue* **THEN**

SETUP: Create a signal so that *si_code* contains the value SI_QUEUE.

TEST: The *si_value* contains the application-specified signal value.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

29 **IF** {_POSIX_REALTIME_SIGNALS} **THEN**

IF *PCTS_timer_settime* and *PCTS_timer_create* **THEN**

SETUP: Create a signal so that *si_code* contains the value SI_TIMER.

TEST: The *si_value* contains the application-specified signal value.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

30 **IF** {_POSIX_ASYNCHRONOUS_IO} **THEN**

SETUP: Create a signal so that *si_code* contains the value SI_ASYNCIO.

TEST: The *si_value* contains the application-specified signal value.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

31 **IF** {_POSIX_MESSAGE_PASSING) **THEN**

SETUP: Create a signal so that *si_code* contains the value SI_MESGQ.

TEST: The *si_value* contains the application-specified signal value.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

D_8 **IF** a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the *context* parameter to a signal-catching function does so in 3.3.1.3.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

D_9 **IF** a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the behavior of a process after it returns normally from a signal-catching function for a SIGFPE, SIGHILL, SIGSEGV, or SIGBUS signal that was not generated by the *kill* () function, the *sigqueue*() function, or the *raise*() function, as defined by the C Standard {2}, does so in 3.3.1.3.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

32 **IF** *PCTS_aio_error* **THEN**
TEST: The function *aio_error*() is reentrant with respect to signals; that is, applications may invoke them, without restriction, from signal-catching functions.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

33 **IF** *PCTS_aio_return* **THEN**
TEST: The function *aio_return*() is reentrant with respect to signals; that is, applications may invoke them, without restriction, from signal-catching functions.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

34 **IF** *PCTS_aio_suspend* **THEN**
TEST: The function *aio_suspend*() is reentrant with respect to signals; that is, applications may invoke them, without restriction, from signal-catching functions.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

35 **IF** *PCTS_clock_gettime* **THEN**
TEST: The function *clock_gettime*() is reentrant with respect to signals; that is, applications may invoke them, without restriction, from signal-catching functions.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

36 **IF** *PCTS_fdatasync* **THEN**
TEST: The function *fdatasync*() is reentrant with respect to signals; that is, applications may invoke them, without restriction, from signal-catching functions.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

37 **IF** *PCTS_sem_post* **THEN**
TEST: The function *sem_post*() is reentrant with respect to signals; that is, applications may invoke them, without restriction, from signal-catching functions.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

38 **IF** *PCTS_sig_queue* **THEN**
TEST: The function *sig_queue*() is reentrant with respect to signals; that is, applications may invoke them, without restriction, from signal-catching functions.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

39 **IF** *PCTS_timer_getoverrun* **THEN**
TEST: The function *timer_getoverrun*() is reentrant with respect to signals; that is, applications may invoke them, without restriction, from signal-catching functions.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

40 **IF** *PCTS_timer_gettime* **THEN**
TEST: The function *timer_gettime*() is reentrant with respect to signals; that is, applications may invoke them, without restriction, from signal-catching functions.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

41 **IF** *PCTS_timer_settime* **THEN**
TEST: The function *timer_settime*() is reentrant with respect to signals; that is, applications may invoke them, without restriction, from signal-catching functions.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

3.3.1.4 Signal Effects on Other Functions

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

NOTE: There are assertions in IEEE Std 2003.1-1992 {4} for this clause that mention specific functions. There may need to be additional assertions here that correspond to those assertions in IEEE Std 2003.1-1992 {4} and cover the new functions of IEEE Std 1003.1b-1993.

3.3.2 Send a Signal to a Process

Function: *kill*().

3.3.2.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.2.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.2.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.2.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.3 Manipulate Signal Sets

Functions: *sigemptyset*(), *sigfillset*(), *sigaddset*(), *sigdelset*(), *sigismember*().

3.3.3.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.3.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.3.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.3.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.4 Examine and Change Signal Action

Function: *sigaction*().

3.3.4.1 Synopsis

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.4.2 Description

1 **SETUP:** Call *sigaction*() with the SA_SIGINFO flag cleared in the *sa_flags* field of the *sigaction* structure to establish a signal-catching function.
TEST: The *sa_handler* field identifies the action to be associated with the specified signal.
TR: Test for all catchable signals consistent with implemented options.
Conformance for sigaction: PASS

2 **IF** {_POSIX_REALTIME_SIGNALS} **THEN**
SETUP: Call *sigaction*() with the SA_SIGINFO flag set in the *sa_flags* field of the *sigaction* structure to establish a signal-catching function.
TEST: The *sa_sigaction* field specifies a signal-catching function.
TR: Test for all catchable signals consistent with implemented options.
ELSE *NO_OPTION*
Conformance for sigaction: PASS, NO_OPTION

3 **SETUP:** Call *sigaction*() with the SA_SIGINFO bit cleared, and the *sa_handler* field specifying a signal-catching function, and then perform the test. Then call it again with the SA_SIGINFO bit set and perform the test.
TEST: The *sa_mask* field identifies a set of signals that are added to the signal mask of the process before the signal-catching function is invoked.
TR: Test for all catchable signals consistent with implemented options. Do not test for SIGKILL and SIGSTOP.
Conformance for sigaction: PASS

4 **SETUP:** Include the header `<signal.h>`.
TEST: The following flag bits are defined and can be set in *sa_flags:*

Symbolic Constant	Description
SA_NOCLDSTOP	Do not generate SIGCHLD when children stop.
SA_SIGINFO	Invoke the signal-catching function with three arguments instead of one.

Conformance for sigaction: PASS

D_1 **TEST:** The PCD.1b documents the disposition of subsequent occurrences of *sig,* when it is already pending, if SA_SIGINFO is not set in *sa_flags*; and does so in 3.3.4.2.
Conformance for sigaction: PASS

5 **IF** {POSIX_REALTIME_SIGNALS} **THEN**
SETUP: Set SA_SIGINFO in *sa_flags* and establish a signal-catching function by calling *sigaction*().
TEST: Subsequent occurrences of *sig* generated by *sigqueue*() or as a result of any signal-generating function that supports the specification of an application-defined value – when *sig* is already pending – is queued in FIFO order until delivered; the signal-catching function is invoked with three arguments; and the application specified value is passed to the signal-catching function as the *si_value* member of the *siginfo_t* structure.
TR: Test for the following functions:

1. If {_POSIX_REALTIME_SIGNALS} is defined then *kill*()
2. If *PCTS_sigqueue* then *sigqueue*()

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3. If *PCTS_lio_listio* then *lio_listio*()

4. If *PCTS_timer_create* and *PCTS_timer_settime* then *timer_settime*()

5. If *PCTS_mq_open* and *PCTS_mq_send* and *PCTS_mq_notify*, then *mq_notify*().

ELSE *NO_OPTION*
Conformance for sigaction: PASS, NO_OPTION

3.3.4.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.4.4 Errors

6 **IF** {POSIX_REALTIME_SIGNALS} is not supported **THEN**
TEST: A call to *sigaction*() with the SA_SIGINFO bit flag set in the *sa_flags* field of the *sigaction* structure returns a value of -1 and sets *errno* to [ENOTSUP].
ELSE *NO_OPTION*
Conformance for sigaction: PASS, NO_OPTION

3.3.5 Examine and Change Blocked Signals

Function: *sigprocmask*().

3.3.5.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.5.2 Description

D_1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of generating a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal while they are blocked, when the signal was not generated by the *kill*() function, the *sigqueue*() function, or the *raise*() function as defined by the C Standard {2} does so in 3.3.5.2.
ELSE *NO_OPTION*
Conformance for sigprcmask: PASS, NO_OPTION

3.3.5.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.5.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.6 Examine Pending Signals

Function: *sigpending*().

3.3.6.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.3.6.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.6.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.6.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.7 Wait for a Signal

Function: *sigsuspend*().

3.3.7.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.7.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.7.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.7.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

3.3.8 Synchronously Accept a Signal

Function: *sigwaitinfo*(), *sigtimedwait*().

3.3.8.1 Synopsis

1

M_GA_stdC_proto_decl(int; sigwaitinfo: const sigset_t *set, siginfo_t *info; signal.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sigwaitinfo: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sigwaitinfo; signal.h;;;)
SEE: Assertion GA_commonc_int_result_decl in 2.7.3.
Conformance for sigwaitinfo: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sigwaitinfo; signal.h;;;)
SEE: Assertion GA_macro_result in 1.3.4.
Conformance for sigwaitinfo: PASS, NO_OPTION

4

M_GA_macro_args(sigwaitinfo; signal.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sigwaitinfo: PASS, NO_OPTION

5

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

M_GA_stdC_proto_decl(int; sigtimedwait; , const sigset_t *set, siginfo_t *info, const struct timespec *timeout; signal.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sigtimedwait: PASS[5, 6], NO_OPTION

6

M_GA_commonC_int_result_decl(sigtimedwait; , const sigset_t *set, siginfo_t *info, const struct timespec *timeout; signal.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sigtimedwait: PASS, NO_OPTION

7

M_GA_macro_result_decl(int; sigtimedwait; signal.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sigtimedwait: PASS, NO_OPTION

8

M_GA_macro_args (sigtimedwait; signal.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sigtimedwait: PASS, NO_OPTION

3.3.8.2 Description

9 **IF** *PCTS_sigwaitinfo* **THEN**
TEST: The function *sigwaitinfo*() waits for the pending signal from the set of signals specified by the *set* parameter and returns the selected signal number.
TR : Test for only one signal at a time by specifying them in *set;* then send two signals, one of which is the signal specified in *set*. Test this for signals in the range SIGRTMIN to SIGRTMAX. Then do the same test separately for all non-realtime signals supported by the implementation.
ELSE *NO_OPTION*
Conformance for sigwaitinfo: PASS, NO_OPTION

10 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**
TEST: When there are multiple pending signals in the range SIGRTMIN to SIGRTMAX, which have been specified in the *set* argument to *function*(), the signal number returned is the lowest numbered one.
NOTE: This can be tested by creating two processes, one to send signals and one to wait for them by calling *function*(). Block the multiple signals to be tested. Then send those signals. Then wait for signals by calling *function*().
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

D_1 FOR: *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* and a PCD.1 documents the following **THEN**
TEST: A PCD.1b that documents the selection order between realtime and nonrealtime signals, or between multiple pending nonrealtime signals, does so in 3.3.8.2.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

11 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**
SETUP: Call *function*() with no signal in *set* pending at the time of the call.
TEST: The process calling *function*() is suspended until one or more signals in *set* become pending (in which case the selected signal number is returned) or until it is interrupted by an unblocked, caught signal.
TR: Test for both cases: when no signals are pending, and when the call is interrupted by an unblocked, caught signal.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

12 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**
SETUP: Call *function*() with the *info* argument non-**NULL**.
TEST: The selected signal number is stored in the *si_signo* member. The cause of the signal is stored in the *si_code* member of the *siginfo_t* structure pointed to by the *info* argument. The selected signal number is returned.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

13 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**
SETUP: Queue different application-specified values for the signal to be selected by a call to *function*() with a non-**NULL** value in the *info* argument.
TEST: The first queued value is dequeued and stored in the *si_value* member of the *siginfo_t* structure pointed to by the *info* argument after a call to *function*() with a non-**NULL** *info* argument. The selected signal number is returned.
TR: Test that all queued values are retrieved in queued order.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

14 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**
TEST: The system resource used to queue the signal selected by *function*() is released and made available to queue other signals when *sigwaitinfo*() returns with the selected signal number.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

D_2 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the content of the *si_value* member of the *info* argument when no value is queued does so in 3.3.8.2.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

15 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**
TEST: After dequeuing all pending signals selected by *function*(), the pending indication for each signal is reset.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

16 **IF** *PCTS_sigtimedwait* **THEN**
SETUP: Call *sigtimedwait*() when none of the signals specified by the *set* arguments are pending.
TEST: The function *sigtimedwait*() waits for the time interval specified in the *timespec* structure referenced by *timeout* before returning.
ELSE *NO_OPTION*
Conformance for sigtimedwait: PASS, NO_OPTION

17 **IF** *PCTS_sigtimedwait* **THEN**
SETUP: Call *sigtimedwait*() with the *timespec* structure pointed to by *timeout* being zero-valued and none of the signals specified by *set* pending.
TEST: The *sigtimedwait*() returns immediately with an error.
ELSE *NO_OPTION*
Conformance for sigtimedwait: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D_3 IF *PCTS_sigtimedwait* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of *sigtimedwait()*, when the *timeout* argument to *sigtimedwait*() is the **NULL** pointer, does so in 3.3.8.2.
ELSE *NO_OPTION*
Conformance for sigtimedwait: PASS, NO_OPTION

18 FOR: *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF ***PCTS_function*** **THEN**
TEST: While *function*() is waiting and a signal occurs that is eligible for delivery (i.e., not blocked by the process signal mask), that signal is handled asynchronously and the wait is interrupted.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

D_4 FOR: *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF not {_POSIX_REALTIME_SIGNALS} and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents its support or lack of support for *function*() does so in 3.3.8.2.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

3.3.8.3 Returns

R_1 TEST: Upon successful completion (i.e., one of the signals specified by *set* is pending or is generated), *sigwaitinfo*() and *sigtimedwait*() return the selected signal number.
SEE: All assertions in 3.3.8.2.

R_2 TEST: The functions *sigwaitinfo*() and *sigtimedwait*() return a value of -1 and set *errno* to indicate the error.
SEE: All assertions in 3.3.8.4.

3.3.8.4 Errors

19 FOR: *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF ***PCTS_function*** **THEN**
SETUP: Call *function*() to wait on an unblocked signal that will be caught.
TEST: The function *function*() returns -1 and sets *errno* to [EINTR] when the wait was interrupted by an unblocked, caught signal.
TR: Test for each catchable signal consistent with the implemented options.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

20 FOR: *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF not *PCTS_function* **THEN**
TEST: A call to *function*() returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

21 IF not *PCTS_sigtimedwait* **THEN**
TEST: A call to *sigtimedwait*() returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sigtimedwait: PASS, NO_OPTION

22 IF *PCTS_sigwaitinfo* and *PCTS_SIGTIMEDWAIT_VALUE* **THEN**
TEST: The *sigtimedwait*() function when called with a timeout argument specifying a *tv_nsec* value less than zero or greater than or equal to 1000 million, and when no signal is pending in *set* and it is necessary to wait, returns -1 and sets *errno* to [EINVAL].
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for sigwaitinfo: PASS, NO_OPTION

D_5 IF *PCTS_sigwaitinfo* and *PCTS_SIGTIMEDWAIT_VALUE* **THEN**
TEST: A PCD.1b that documents the detection of a *timeout* argument specifying a *tv_nsec* value less than zero, or greater than or equal to 1000 million, and the conditions under which the error is detected, does so in 3.3.8.4.
ELSE *NO_OPTION*
Conformance for sigwaitinfo: PASS, NO_OPTION

3.3.9 Queue a Signal to a Process

Function: *sigqueue*().

3.3.9.1 Synopsis

1
M_GA_stdC_proto_decl(int; sigqueue; pid_t pid, int signo, const union sigval value; signal.h;;;)
SEE: Assertion GA_stC_proto_decl in 2.7.3.
Conformance for sigqueue: PASS[1,2], NO_OPTION

2
MG_GA_commonC_int_result_decl(sigqueue; signal.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sigqueue: PASS[1,2], NO_OPTION

3
MG_GA_macro_result_decl(int; sigqueue; signal.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sigqueue: PASS, NO_OPTION

4 ***MG_GA_macro_args(sigqueue; signal.h;;;)***
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sigqueue: PASS, NO_OPTION

3.3.9.2 Description

sigqueue_priv()=
Create two processes, one to send the signal and one to receive it, such that if *PCTS_GAP_sigqueue*, the real or effective user ID of the calling process, does not match the real or effective user ID of the process to which the signal is being sent, or if {_POSIX_SAVED_IDS} is the real or effective user ID of the calling process matches the real or saved set-user-ID of the process to which the signal is being sent; otherwise, the real or effective user ID of the calling process matches the real or effective user ID of the process to which the signal is being sent.

5 **IF** *PCTS_sigqueue* **THEN**
SETUP:
***sigqueue_priv*()**
TEST: The *sigqueue*() function causes the signal specified by *signo* to be sent with the value specified by *value* to the process specified by *pid.*
TR: Test for all catchable signals consistent with implemented options. Also, test for the supported combinations of real and effective user IDs as well as for appropriate privilege and saved set-user-IDs, if they are supported by the implementation.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6 **IF** *PCTS_sigqueue* **THEN**
IF {SIGQUEUE_MAX} <= *PCTS_SIGQUEUE_MAX* **THEN**
SETUP:
sigqueue_priv()
TEST: The *sigqueue*() function can queue a total of {SIGQUEUE_MAX} signals to one or more processes.
TR: Test for one and two processes.

Test for all catchable signals consistent with implemented options. Also, test for the supported combinations of real and effective user IDs as well as for appropriate privilege and saved set-user-IDs, if they are supported by the implementation.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_sigqueue* **THEN**
IF {SIGQUEUE_MAX} > *PCTS_SIGQUEUE_MAX* **THEN**
SETUP:
sigqueue_priv()
TEST: The *sigqueue*() function can queue a total of *PCTS_SIGQUEUE_MAX* signals to one or more processes.
TR: Test for one and two processes.

Test for all catchable signals consistent with implemented options. Also, test for the supported combinations of real and effective user IDs as well as for appropriate privilege and saved set-user-IDs, if they are supported by the implementation.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_sigqueue* **THEN**
SETUP:
sigqueue_priv()
TEST: If the *signo* argument in *sigqueue*() is zero (the null signal), error checking is performed but no signal is actually sent.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_OPTION

9 **IF** *PCTS_sigqueue* **THEN**
SETUP:
sigqueue_priv()
Block the signal specified in *signo* for the receiving process and clear the SA_SIGINFO flag for it.
TEST: A call to *sigqueue*() returns immediately.
TR: Test for all catchable signals consistent with implemented options. Also, test for the supported combinations of real and effective user IDs, as well as for appropriate privilege and saved set-user-IDs, if they are supported by the implementation.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_OPTION

10 **IF** *PCTS_sigqueue* **THEN**
SETUP:
sigqueue_priv()
Block the signal specified in *signo* for the receiving process and set the SA_SIGINFO flag for it.
TEST: A call to *sigqueue*() returns immediately, and if the resources are available to queue the signal, the signal is left queued and pending.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TR: Test for all catchable signals consistent with the implemented options. Also, test for the supported combinations of real and effective user IDs, as well as for appropriate privilege and saved set-user-IDS, if they are supported by the implementation.

ELSE *NO_OPTION*

Conformance for sigqueue: PASS, NO_OPTION

11 **IF** *PCTS_sigqueue* **THEN**

SETUP:

sigqueue_priv()

Do not set SA_SIGINFO for *signo*

TEST: The *signo* is sent at least once to the receiving process.

TR: Test for all catchable signals consistent with the implemented options. Also, test for the supported combinations of real and effective user IDs, as well as for appropriate privilege and saved set-user-IDs, if they are supported by the implementation.

ELSE *NO_OPTION*

Conformance for sigqueue: PASS, NO_OPTION

D_1 **IF** *PCTS_sigqueue* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not the *value* parameter is sent to the receiving process as a result of calling the *sigqueue*() function when SA_SIGINFO is not set for *signo,* and the conditions under which the *value* is sent, does so in 3.3.9.2.

ELSE *NO_OPTION*

Conformance for sigqueue: PASS, NO_OPTION

12 **IF** *PCTS_sigqueue* **THEN**

SETUP:

Set *pid* so that it causes *signo* to be generated for the sending process, and do not block *signo.*

TEST: Either *signo*, or at least the pending, unblocked signal with the lowest number, is delivered to the sending process before the *sigqueue*() function returns.

TR: Test for all catchable signals consistent with the implemented options. Also, test for the supported combinations of real and effective user IDs, as well as for appropriate privilege and saved set-user-IDs, if they are supported by the implementation.

ELSE *NO_OPTION*

Conformance for sigqueue: PASS, NO_OPTION

D-2 **IF** not {_POSIX_REALTIME_SIGNALS} and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents its support or lack of support for *sigqueue*() does so in 3.3.8.2.

ELSE *NO_OPTION*

Conformance for sigqueue: PASS, NO_OPTION

3.3.9.3 Returns

R-1 **IF** *PCTS_sigqueue* **THEN**

TEST: Upon successful completion of a call to *sigqueue*(), the specified signal has been queued, and the function returns a value of zero.

ELSE *NO_OPTION*

SEE: Assertions in 3.3.9.2.

R-2 **IF** *PCTS_sigqueue* **THEN**

TEST: An unsuccessful call to *sigqueue*() returns a value of -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: Assertions in 3.3.9.4.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.3.9.4 Errors

13 **IF** *PCTS_sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [EAGAIN] when no resources are available to queue the signal.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST, NO_OPTION

14 **IF** *PCTS_sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [EAGAIN] when the process has already queued {SIGQUEUE_MAX} signals that are still pending at the receiver(s).
NOTE: There is no known portable test method for this assertion.
TR: Test limit for signals sent to one process and to two processes.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_OPTION

15 **IF** *PCTS_sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [EAGAIN] when a systemwide resource limit has been exceeded.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST, NO_OPTION

16 **IF** *PCTS_sigqueue* **THEN**
IF *PCTS_INVALID_SIGNAL* or *PCTS_UNSUPPORTED_SIGNAL* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [EINVAL] when the value of the *signo* argument is an invalid or unsupported signal number.
TR: Test for both invalid and unsupported signals, if each exists.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST_SUPPORT, NO_OPTION

17 **IF** not *PCTS_sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [ENOSYS] when the function *sigqueue*() is not supported by this implementation.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_OPTION

18 **IF** *PCTS_sigqueue* **THEN**
IF ***PCTS_RAP***_*sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [EPERM] when the process does not have the appropriate privilege to send the signal to the receiving process.
TR: Use *sigqueue_priv*().
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [ESDRCH] when the process *pid* does not exist.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.4 Timer Operations

There are no requirements for conforming implementations in this clause.

3.4.1 Schedule Alarm

Function: *alarm*().

3.4.1.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

3.4.1.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

3.4.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

3.4.1.4 Errors

There are no requirements for conforming implementations in this clause.

3.4.2 Suspend Process Execution

Function: *pause*().

3.4.2.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

3.4.2.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

3.4.2.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

3.4.2.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

3.4.3 Delay Process Execution

Function: *sleep*().

3.4.3.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

3.4.3.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.4.3.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

3.4.3.4 Errors

There are no requirements for conforming implementations in this clause.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Section 4: Process Environment

There are no POSIX.1b {3} assertions in Section 4, except for 4.8.1.2.

4.8 Configurable System Variables

4.8.1 Get Configurable System Variables

Function: *sysconf*().

4.8.1.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

4.8.1.2 Description

NOTE: There is no Table 4-1 in this section. Table 4-2 in this standard is the same as Table 4-2 in POSIX.1b {3}.

(IEEE Std 2003.1-1992 {4} 04
UNUSED

1 **SETUP:** Include the header `<unistd.h>`.
TEST: The symbolic constants in the *name* Value column of Table 4-2 are defined and have different values.
NOTE: The assertion is tested once for each constant specified in the FOR clause. The assertion is to be read by substituting CONSTANT with the current constant specified in the FOR clause. The name of the compound constant also is to be substituted for each occurrence in the constructs *_SC_CONSTANT* and *_POSIX_CONSTANT*.
Conformance for sysconf: PASS

2 **FOR:** {AIO_LISTIO_MAX}, {AIO_MAX}, {AIO_PRIO_DELTA_MAX}, {DELAYTIMER_MAX}, {MQ_OPEN_MAX}, {MQ_PRIO_MAX}, {PAGESIZE}, {RTSIG_MAX}, {SEM_NSEMS_MAX}, {SEM_VALUE_MAX}, {SIGQUEUE_MAX}, {TIMER_MAX},
IF CONSTANT is defined in `<limits.h>` **THEN**
SETUP: Include the header `<limits.h>`.
TEST: A call *sysconf(_SC_CONSTANT)* either returns -1 without changing the value of *errno* or returns a value greater than or equal to {CONSTANT}.
NOTE: The assertion is tested once for each constant specified in the FOR clause. The assertion is to be read by substituting CONSTANT with the current constant specified in the FOR clause. The name of the compound constant also is to be substituted for each occurrence in the constructs *_SC_CONSTANT* and *_POSIX_CONSTANT*.
ELSE *NO_TEST_SUPPORT*
Conformance for sysconf: PASS, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3 **FOR:** {AIO_LISTIO_MAX}, {AIO_MAX}, {AIO_PRIO_DELTA_MAX}, {DELAYTIMER_MAX}, {MQ_OPEN_MAX}, {MQ_PRIO_MAX}, {PAGESIZE}, {RTSIG_MAX}, {SEM_NSEMS_MAX}, {SEM_VALUE_MAX}, {SIGQUEUE_MAX}, {TIMER_MAX},

IF CONSTANT is **not** defined in `<limits.h>` **THEN**

SETUP: Include the header `<limits.h>`.

TEST: A call *sysconf(_SC_CONSTANT)* either returns -1 without changing the value of *errno* or returns a value greater than or equal to {_POSIX_CONSTANT}.

NOTE: The assertion is tested once for each constant specified in the FOR clause. The assertion is to be read by substituting CONSTANT with the current constant specified in the FOR clause. The name of the compound constant also is to be substituted for each occurrence in the constructs *_SC_CONSTANT* and *_POSIX_CONSTANT*.

ELSE *NO_TEST_SUPPORT*

Conformance for sysconf: PASS, NO_TEST_SUPPORT

4 **FOR:** {_POSIX_ASYNCHRONOUS_IO}, {_POSIX_FSYNC}, {_POSIX_MAPPED_FILES}, {_POSIX_MEMLOCK}, {_POSIX_MEMLOCK_RANGE}, {_POSIX_MEMORY_PROTECTION}, {_POSIX_MESSAGE_PASSING}, {_POSIX_PRIORITIZED_IO}, {_POSIX_PRIORITY_SCHEDULING}, {_POSIX_REALTIME_SIGNALS}, {_POSIX_SEMAPHORES}, {_POSIX_SHARED_MEMORY_OBJECTS}, {POSIX_SYNCHRONIZED_IO}, {_POSIX_TIMERS},

IF CONSTANT is defined in `<unistd.h>` **THEN**

SETUP: Include the header `<unistd.h>`.

TEST: A call *sysconf(_SC_CONSTANT)* returns a value other than -1.

NOTE: The assertion is tested once for each constant specified in the FOR clause. The assertion is to be read by substituting CONSTANT with the current constant specified in the FOR clause. The name of the compound constant also is to be substituted for each occurrence in the constructs *_SC_CONSTANT* and *_POSIX_CONSTANT*.

ELSE *NO_TEST_SUPPORT*

Conformance for sysconf: PASS, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 4-2 – Configurable System Variables

Variable	*name* Value
{AIO_LISTIO_MAX}	{_SC_AIO_LISTIO_MAX}
{AIO_MAX}	{_SC_AIO_MAX}
{AIO_PRIO_DELTA_MAX}	{_SC_AIO_PRIO_DELTA_MAX}
{ART_MAX}	{_SC_ARG_MAX}
{CHILD_MAX}	{_SC_CHILD_MAX}
clock ticks/second	{_SC_CLK_TCK}
{DELAYTIMER_MAX}	{_SC_DELAYTIMER_MAX}
{MQ_OPEN_MAX}	{_SC_MQ_OPEN_MAX}
{MQ_PRIO_MAX}	{_SC_MQ_PRIO_MAX}
{NGROUPS_MAX}	{_SC_NGROUPS_MAX}
{OPEN_MAX}	{_SC_OPEN_MAX}
{PAGESIZE}	{_SC_PAGESIZE}
{RTSIG_MAX}	{_SC_RTSIG_MAX}
{SEM_NSEMS_MAX}	{_SC_SEM_NSEMS_MAX}
{SEM_VALUE_MAX}	{_SC_SEM_VALUE_MAX}
{SIGQUEUE_MAX}	{_SC_SIGQUEUE_MAX}
{STREAM_MAX}	{_SC_STREAM_MAX}
{TIMER_MAX}	{_SC_TIMER_MAX}
{TZNAME__MAX}	{_SC_TZNAME_MAX}
{_POSIX_ASYNCHRONOUS_IO}	{_SC_ASYNCHRONOUS_IO}
{_POSIX_FSYNC}	{_SC_FSYNC}
{_POSIX_JOB_CONTROL}	{_SC_JOB_CONTROL}
{_POSIX_MAPPED_FILES}	{_SC_MAPPED_FILES}
{_POSIX_MEMLOCK}	{_SC_MEMLOCK}
{_POSIX_MEMLOCK_RANGE}	{_SC_MEMLOCK_RANGE}
{_POSIX_MEMORY_PROTECTION}	{_SC_MEMORY_PROTECTION}
{_POSIX_MESSAGE_PASSING}	{_SC_MESSAGE_PASSING}
{_POSIX_PRIORITIZED_IO}	{_SC_PRIORITIZED_IO}
{_POSIX_PRIORITY_SCHEDULING}	{_SC_PRIORITY_SCHEDULING}
{_POSIX_REALTIME_SIGNALS}	{_SC_REALTIME_SIGNALS}
{_POSIX_SAVED_IDS}	{_SC_SAVED_IDS}
{_POSIX_SEMAPHORES}	{_SC_SEMAPHORES}
{_POSIX_SHARED_MEMORY_OBJECTS}	{_SC_SHARED_MEMORY_OBJECTS}
{_POSIX_SYNCHRONIZED_IO}	{_SC_SYNCHRONIZED_IO}
{_POSIX_TIMERS}	{_SC_TIMERS}
{_POSIX_VERSION}	{_SC_VERSION}

5 **FOR:** {_POSIX_ASYNCHRONOUS_IO}, {_POSIX_FSYNC}, {_POSIX_MAPPED_FILES}, {_POSIX_MEMLOCK}, {_POSIX_MEMLOCK_RANGE}, {_POSIX_MEMORY_PROTECTION}, {_POSIX_MESSAGE_PASSING}, {_POSIX_PRIORITIZED_IO}, {_POSIX_PRIORITY_SCHEDULING}, {_POSIX_REALTIME_SIGNALS}, {_POSIX_SEMAPHORES}, {_POSIX_SHARED_MEMORY_OBJECTS}, {POSIX_SYNCHRONIZED_IO}, {_POSIX_TIMERS},

IF CONSTANT is **not** defined in `<unistd.h>` **THEN**

SETUP: Include the header `<unistd.h>`.

TEST: A call *sysconf(_SC_CONSTANT)* returns -1 without changing the value of *errno*.

NOTE: The assertion is tested once for each constant specified in the FOR clause. The assertion is to be read by substituting CONSTANT with the current constant specified in the FOR clause. The name of the compound constant also is to be

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

substituted for each occurrence in the constructs *_SC_CONSTANT* and *_POSIX_CONSTANT*.

ELSE *NO_TEST_SUPPORT*
Conformance for sysconf: PASS, NO_TEST_SUPPORT

4.8.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

4.8.1.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

4.8.1.5 Special Symbol {CLK_TCK}

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b (3) assertions.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Section 5: Files and Directories

There are no requirements for conforming implementations in this clause.

5.1 Directories

5.1.1 Format of Directory Entries

There are no requirements for conforming implementations in this clause.

5.1.2 Directory Operations

Functions: *opendir*(), *readdir*(), *rewinddir*(), *closedir*().

5.1.2.1 Synopsis

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.1.2.2 Description

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.1.2.3 Returns

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.1.2.4 Errors

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.2 Working Directory

5.2.1 Change Current Working Directory

Function: *chdir*()

5.2.1.1 Synopsis

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.2.1.2 Description

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5.2.1.3 Returns

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.2.1.4 Errors

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.2.2 Get Working Directory Pathname

Function: *getcwd*().

5.2.2.1 Synopsis

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.2.2.2 Description

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.2.2.3 Returns

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.2.2.4 Errors

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.3 General File Creation

5.3.1 Open a File

Function: *open*().

5.3.1.1 Synopsis

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.3.1.2 Description

NOTE: The General Assertion *GA_syncIODataIntegrityWrite* is defined in 2.2.2.119. It is tested in *write*(), *aio_write*(), and *lio_listio*(), instead of in *open*(), because they are more appropriate places to test it. The General Assertion *GA_syncIODataIntegrityRead* is defined in 2.2.2.119. It is tested in *real*(), *aio_read*(), and *lio_listio*(), instead of in *open*(), because they are more appropriate places to test it. The General Assertion *GA_syncIOFileIntegrityRead* is defined in 2.2.1.120. It is tested in *read*(), *aio_read*(), and *lio-listio*(), instead of in *open*(), because they are more appropriate places to test it. The General Assertion *GA_syncIODataIntegrityWrite* is defined in 2.2.2.120. It is tested in *write*(), *aio_write*(), and *listio*(), instead of in *open*(), because they are more appropriate places to test it.

Also, the O_SYSYNC, O_RSYNC, and O_SYNCH constants are tested for existence and value in 6.5.1.1.

GA_syncOpenWrite

FOR: *write*(), *aio_write*(), and *lio-listio*()

IF {_POSIX_SYNCHRONIZED_IO} **THEN**

SETUP: Open a file by calling *open*() with both O_SYNCH and O_DSYNC set in the *oflag* parameter.

TEST: The file behaves as if only the O_SYNC flag was set.

TR: Test for regular files.

NOTE: There is no known portable test method for this assertion.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
Conformance for open: PASS, NO_TEST_NO OPTION

5.3.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.3.1.4 Errors

1 **IF** {_POSIX_SYNCHRONIZED_IO} **THEN**
IF *PCTS_NO_SYNC_IO_FILE* **THEN**
SETUP: Call *open*()(*path, oflag | oflag1*).
TEST: The *open*() returns -1 and sets *errno* to [EINVAL].
NOTE: This implementation does not support synchronized I/O for this file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for open: PASS, NO_TEST_SUPPORT, NO_OPTION

5.3.1.5 Errors

2 **IF** {_POSIX_SYNCHRONIZED_IO} **THEN**
IF *PCTS_NO_SYNC_IO_FILE* **THEN**
SETUP: Call *open*()(*path, oflag | oflag1*).
TEST: The *open*() returns -1 and sets *errno* to [EINVAL].
NOTE: This implementation does not support synchronized I/O for this file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for open: PASS, NO_TEST_SUPPORT, NO_OPTION

5.3.2 Create New File or Rewrite an Existing One

Function: *creat*().

5.3.2.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.3.2.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.3.3 Set File Creation Mask

Function: *umask*().

5.3.3.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.3.3.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.3.3.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.3.3.4 Errors

There are no requirements for conforming implementations in this clause.

5.3.4 Link to a File

Function: *link*().

5.3.4.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.3.4.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.3.4.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.3.4.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.4 Special File Creation

5.4.1 Make a Directory

Function: *mkdir*().

5.4.1.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.4.1.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.4.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.4.1.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.4.2 Make a FIFO Special File

Function: *mkfifo*().

5.4.2.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.4.2.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5.4.2.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.4.2.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5 File Removal

5.5.1 Remove Directory Entries

Function: *unlink*().

5.5.1.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5.1.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5.1.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5.2 Remove a Directory

Function: *rmdir*().

5.5.2.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5.2.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5.2.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5.2.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5.3 Rename a File

Function: *rename*().

5.5.3.1 Synopsis

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5.3.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5.3.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.5.3.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.6 File Characteristics

5.6.1 File Characteristics: Header and Data Structure

`<sys/stat.h>`

There are only IEEE Std 2003.1-1992 {4} assertions in this clause: there are no POSIX.1b {3} assertions.

5.6.1.1 `<sys/stat.h>` File Types

D_1 If a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether message queues, semaphores, or shared memory objects are implemented as distinct file types, and does so in 5.6.1.1.
ELSE *NO_OPTION*
Conformance for stat.h: PASS, NO_OPTION

1 **SETUP:** Include the header `<sys/stat.h>`.
TEST: The macros *S_TYPEISMQ*(), *S_TYPEISSEM*(), and *S_TYPEISSHM*() are defined.
Conformance for stat.h: PASS

2 **IF** *PCTS_MQ_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Also, create a message queue and put its information into the *stat* structure referenced by the *buf* parameter.
TEST: The macro call *S_TYPEISMQ(buf)* evaluates to a nonzero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

3 **IF** *PCTS_MQ_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a message queue type file into the *stat* structure referenced by the *buf* parameter.
TEST: The macro call *S_TYPEISMQ(buf)* evaluates to a zero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

4 **IF** Not *PCTS_MQ_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a message queue type file into the *stat* structure referenced by the *buf* parameter.
TEST: The macro call *S_TYPEISMQ(buf)* evaluates to a zero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

5 **IF** *PCTS_SEM_IS_FD* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SETUP: Include the header `<sys/stat.h>`. Also, create a semaphore and put its information into the *stat* structure referenced by the *buf* parameter.
TEST: The macro call *S_TYPEISSEM(buf)* evaluates to a nonzero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

6 **IF** *PCTS_SEM_IS_FD* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a semaphore type file into the *stat* structure referenced by the *buf* parameter.
TEST: The macro call *S_TYPEISSEM(buf)* evaluates to a zero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

7 **IF** *PCTS_SEM_IS_FD* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a semaphore type file into the *stat* structure referenced by the *buf* parameter.
TEST: The macro call *S_TYPEISSEM(buf)* evaluates to a zero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

8 **IF** *PCTS_SHM_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Also, create a shared memory object and put its information into the *stat* structure referenced by the *buf* parameter.
TEST: The macro call *S_TYPEISSHM(buf)* evaluates to a nonzero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

9 **IF** *PCTS_SHM_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a shared memory object type file into the *stat* structure referenced by the *buf* parameter.
TEST: The macro call *S_TYPEISSHM(buf)* evaluates to a zero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

10 **IF** *PCTS_SHM_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a shared memory object type file into the *stat* structure referenced by the *buf* parameter.
TEST: The macro call *S_TYPEISSHM(buf)* evaluates to a nonzero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

5.6.1.2 `<sys/stat.h>` File Modes

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.1.3 `<sys/stat.h>` Time Entries

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.2 Get File Status

Functions: *stat*(), *fstat*().

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.6.2.1 Synopsis

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.2.2 Description

1 **SETUP:** Open a shared memory object using *shm_open*(). Call *fstat*() with the *fildes* parameter referring to that shared memory object.
TEST: The *stat* structure pointed to by the *buf* argument has the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits updated correctly.
Conformance for fstat: PASS

5.6.2.3 Returns

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.2.4 Errors

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.3 Check File Accessibility

Function: *access*().

5.6.3.1 Synopsis

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.3.2 Description

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.3.3 Returns

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.3.4 Errors

There are only IEEE Std 2003.1 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.4 Change File Modes

Function: *chmod*(), *fchmod* ().

5.6.4.1 Synopsis

1

M_GA_stdC_proto_decl(int;fchmod;,intfildes, mode_t mode; sys/stat.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3
Conformance for fchmod: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(fchmod; sys/stat.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3
Conformance for fchmod: PASS[1, 2], NO_OPTION

3

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

M_GA_macro_result_decl(int;fchmod; sys/stat.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4
Conformance for fchmod: PASS, NO_OPTION

4

M_GA_macro_args(fchmod; sys/stat.h;;;)
SEE: Assertion GA_macro_macro_args in 2.7.3
Conformance for fchmod: PASS, NO_OPTION

5.6.4.2 Description

5 **IF** *PCTSfchmod* **THEN**
SETUP: Open a file where the effective user ID of the calling process matches the file owner.
TEST: A call *fchmod(fildes, mode*) sets the file permission bits from the corresponding bits in the *mode* argument.
TR: For regular files: test all file permission bits.
For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_OPTION

6 **IF** *PCTS_fchmod* **THEN**
IF *PCTS_GAP_MODES_fchmod* **THEN**
SETUP: Open a file where the effective user ID of the calling process does not match the file owner, and acquire the appropriate privilege to change the file permission bits of the file using *fchmod*().
TEST: A call *fchmod(fildes, mode)* sets the file permission bits from the corresponding bits in the *mode* argument.
TR : For regular files: test all file permission bits.

For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fchmod:PASS, NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_fchmod* and *PCTS_CHMOD_SUID* **THEN**
SETUP: Open a file where the effective user ID of the calling process matches the file owner.
TEST: A call *fchmod(fildes, mode)* sets the S_ISUID bit from the corresponding bit in the *mode* argument.
TR: For regular files: test all file permission bits.
For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.
ELSE *NO_OPTION*
Conformance for fchmod:PASS, NO_OPTION

8 **IF** *PCTS_fchmod* and *PCTS_CHMOD_SUID* **THEN**
IF *PCTS_GAP_SUID_fchmod* **THEN**
SETUP: Open a file where the effective user ID of the calling process does not match the file owner, and acquire the appropriate privilege to change the file permission bits of the file using *fchmod*().
TEST: A call *fchmod(fildes, mode)* sets the S_ISUID bit from the corresponding bit in the *mode* argument.
TR: For regular files: test all file permission bits.
For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fchmod:PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_fchmod* and *PCTS_CHMOD_SGID* **THEN**
SETUP: Open a file where the effective user ID of the calling process matches the file owner.
TEST: A call *fchmod(fildes, mode)* sets the S_ISGID bit from the corresponding bit in the *mode* argument.
TR: For regular files: test all file permission bits.
For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.
ELSE *NO_OPTION*
Conformance for fchmod:PASS, NO_OPTION

10 **IF** *PCTS_fchmod* and *PCTS_CHMOD_SGID* **THEN**
IF *PCTS_GAP_SGID_fchmod* **THEN**
SETUP: Open a file where the effective user ID of the calling process does not match the file owner, and acquire the appropriate privilege to change the file permission bits of the file using *fchmod*().
TEST: A call *fchmod(fildes, mode)* sets the S_ISGID bit from the corresponding bit in the *mode* argument.
TR: For regular files: test all file permission bits.

For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fchmod:PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_fchmod* **THEN**
IF *PCTS_RAP_SGID_fchmod* **THEN**
SETUP: Open a regular file such that the group ID of the file does not match the effective group ID or one of the supplementary group IDs of the process, and one or more of the S_IXUSR, S_IXGRP, or S_IXOTH bits of the file mode are set. Release appropriate privileges to change the S_ISGID bit in the mode of the file.
TEST: Bit S_ISGID (set group ID on execution) in the mode of the file is cleared upon successful return from *fchmod*().
TR: Test for each of the following bits being set individually in the file mode: S_IXUSR, S_IXGRP, and S_IXOTH.

ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fchmod:PASS, NO_TEST_SUPPORT, NO_OPTION

12 **IF** *PCTS_RAP_SGID_chmod* **THEN**
SETUP: Open a regular such that the group ID of the file does not match the effective group ID, or one of the supplementary group IDs of the process and one or more of the S_IXUSR, S_IXGRP, or S_IXOTH bits of the file mode are set. Release appropriate privileges to change the S_ISGID bit in the mode of the file.
TEST: Bit S_ISGID (set group ID on execution) in the mode of the file is cleared upon successful return from *chmod*().
TR: Test for each of the following bits being set individually in the file mode: S_IXUSR, S_IXGRP, and S_IXOTH.
ELSE *NO_TEST_SUPPORT*
Conformance for chmod:PASS, NO_TEST_SUPPORT,

13 **IF** *PCTS_fchmod* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: Upon successful completion, the *fchmod*() function marks for update the *st_ctime* field of the file.
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_OPTION

5.6.4.3 Returns

R_1 IF *PCTS_fchmod* **THEN**
TEST: Upon successful completion, the *fchmod*() function returns a value of zero.
ELSE *NO_OPTION*
SEE: Assertions in 5.6.4.2.

R_2 IF *PCTS_fchmod* **THEN**
TEST: Upon an unsuccessful completion, the *fchmod*() function returns a value of -1, sets *errno* to indicate the error, and no change to the file *mode* occurs.
ELSE *NO_OPTION*
SEE: Assertions in 5.6.4.4.

5.6.4.4 Errors

14 **IF** *PCTS_fchmod* **THEN**
SETUP: Call *fchmod*() with a *fildes* argument that is not a valid file descriptor.
TEST: A call to the *fchmod*() function on such a *fildes* returns a value of -1, sets *errno* to [EBADF], and no change to the file *mode* occurs.
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_OPTION

15 **IF** not *PCTS_fchmod* **THEN**
TEST: A call to the *fchmod*() function on such a *fildes* returns a value of -1, sets *errno* to [ENOSYS], and no change to the file *mode* occurs.
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_OPTION

16 **IF** *PCTS_fchmod* **THEN**
SETUP: The effective user ID does not match the owner of the file, and the calling process does not have the appropriate privileges.
TEST: A call to the *fchmod*() function on such a *fildes* returns a value of -1, sets *errno* to [EPERM], and no change to the file *mode* occurs.
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_OPTION

17 **IF** *PCTS_fchmod* **THEN**
IF *PCTS_ROFS* **THEN**
SETUP: Open a file that resides on a read-only file system.
TEST: A call to the *fchmod*() function on such a *fildes* returns a value of -1, sets *errno* to [EROFS], and no change to the file *mode* occurs.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_fchmod* and *PCTS_EINVAL_fchmod* **THEN**
SETUP: Open a file such that the *fildes* argument refers to a pipe.
TEST: A call to the *fchmod*() function on such a *fildes* returns a value of -1, sets *errno* to [EINVAL], and no change to the file *mode* occurs.
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.6.5 Change Owner and Group of a File

Function: *chown*().

5.6.5.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.5.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.5.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.5.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.6 Set File Access and Modification Times

Function: *utime*().

5.6.6.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.6.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.6.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.6.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.6.7 Truncate a File to a Specified Length

Function: *ftruncate*().

1

M_GA_stdC_proto_decl(int; ftruncate; int fildes, off_t length; unistd.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3
Conformance for ftruncate: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(ftruncate; unistd.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3
Conformance for ftruncate: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; ftruncate; unistd.h;;;)
SEE: Assertion GA_macro__result_decl in 1.3.4

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for ftruncate: PASS, NO_OPTION

4

M_GA_macro_args(ftruncate; unistd.h;;;)
SEE: Assertion GA_macro__args in 2.7.3
Conformance for ftruncate: PASS, NO_OPTION

5.6.7.2 Description

5 **IF** *PCTS_ftruncate* **THEN**
SETUP: Open a regular file where the size of each file exceeds the *length* specified in the *ftruncate*() call to be tested.
TEST: The call *ftruncate(fildes, length)* causes the regular file known by the file descriptor *fildes* to be truncated to *length*.

TR: Test for write-only and read-write access to the file.
ELSE *NO_OPTION*
Conformance for fruncate: PASS, NO_OPTION

D_1 **IF** *PCTS_ftruncate* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the file is changed, or its size increased after a call *ftruncate(fildes, length)* if the file previously was smaller than *length*, does so in 5.6.7.2.
ELSE *NO_OPTION*
Conformance for fruncate: PASS, NO_OPTION

6 **IF** *PCTS_ftruncate* **THEN**
IF *PCTS_EXTEND_ON_ftruncate* **THEN**
SETUP: Open a regular file whose size is less than the size to which it will be truncated.
TEST: The call *ftruncate(fildes, length),* where length is greater than the size of the file, causes the extended area to appear as if it were zero-filled.
TR: Test for write-only and read-write access to the file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fruncate: PASS, NO_TEST_SUPPORT_,NO_OPTION

7 **IF** *PCTS_ftruncate* and *PCTS_shm_open* **THEN**
TEST: A call *ftruncate(fildes, length),* where *fildes* references a shared memory object, sets the size of the shared memory object to *length*.
TR: Test for shared memory objects both larger and smaller than *length*.
ELSE *NO_OPTION*
Conformance for fruncate: PASS, NO_OPTION

D_2 **IF** *PCTS_ftruncate* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of calling *ftruncate*() on a file that is not a regular file or a shared memory object does so in 5.6.7.2.
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8 **IF** *PCTS_ftruncate* and *PCTS_mmap* **THEN**
IF *_POSIX_MEMORY_PROTECTION* **THEN**
SETUP: Memory map a file so that the effect of *ftruncate*() is to decrease the size of a file and whole pages beyond the new end were previously mapped.
TEST: The whole mapped pages beyond the new end after a call to *ftruncate*() are discarded and references to them result in delivery of a SIGBUS signal.

ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_ftruncate* and PCTS_shm_open and *PCTS_mmap* **THEN**
IF *_POSIX_MEMORY_PROTECTION* **THEN**
SETUP: Memory map a shared memory object so that the effect of *ftruncate*() is to decrease the size of the shared memory object and whole pages beyond the new end were previously mapped.
TEST: The whole mapped pages beyond the new end after a call to *ftruncate*() are discarded and references to them result in delivery of a SIGBUS signal.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_TEST_SUPPORT, NO_OPTION

10 **IF** *PCTS_ftruncate* **THEN**
TEST: The value of the seek pointer is not to be modified by a ll to *ftruncate*().
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

11 **IF** *PCTS_ftruncate* **THEN**
TEST: Upon successful completion, the *ftruncate*() function marks for update the *st_ctime* and *st_mtime* fields of the file.
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

12 **IF** *PCTS_ftruncate* **THEN**
TEST: The file is unaffected by an unsuccessful call to *ftruncate*().
TR: Test the size of the file, as well as the *st_ctime* and *st_mtime* fields of the file.
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

5.6.7.3 Returns

R_1 **IF** *PCTS_ftruncate* **THEN**
TEST: Upon successful completion, the *ftruncate*() function returns to zero.
ELSE *NO_OPTION*
SEE: Assertions in 5.6.7.2.

R_2 **IF** *PCTS_ftruncate* **THEN**
TEST: An unsuccessful call to the *ftruncate*() function returns -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: Assertions in 5.6.7.4.

5.6.7.4 Errors

13 **IF** *PCTS_ftruncate* **THEN**
TEST: A call to *ftruncate*(), where the *fildes* argument is not a valid file descriptor open for writing, returns -1 and sets *errno* to [EBADF].
TR: Test the size of the file, as well as the *st_ctime* and *st_mtime* fields of the file.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

14 **IF** *PCTS_ftruncate* **THEN**
TEST: A call to *ftruncate*(), where the *fildes* argument does not refer to a file on which this operation is possible, returns -1 and sets *errno* to [EINVAL]
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

15 **IF** *PCTS_ftruncate* **THEN**
TEST: A call to *ftruncate*(), where the file resides on a read-only file system, returns -1 and sets *errno* to [EROFS]
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

5.7 Configurable Pathname Variables

5.7.1 Get Configurable Pathname Variables

Functions: *pathconf*(), *fpathconf*().

5.7.1.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.7.1.2 Description

1 **SETUP:** Include the header <unistd.h>.
TEST: The constants {_PC_ASYNC_IO} and {_PC_SYNC_IO} are defined.
Conformance for pathconf: PASS

2 **FOR:** *pathconf*(), and *fpathconf*()
IF {_POSIX_ASYNCH_IO} is defined when <unistd.h> is included **THEN**
SETUP: Include the header <unistd.h>
TEST: A call *function()* with a name parameter equal to {_POSIX_ASYNCH_IO} returns a value equal to {_POSIX_ASYNCH_IO}.
ELSE *NO_OPTION*
Conformance for pathconf, fpathconf: PASS, NO_OPTION

3 **FOR:** *pathconf*() and *fpathconf*()
SETUP: Include the header <unistd.h>
TEST: A call *function*() with a *name* parameter equal to {_POSIX_ASYNCH_IO} that refers to a file other than a directory returns a value corresponding to the option {_POSIX_ASYNCH_IO} for that file.
Conformance for pathconf, fpathconf: PASS

4 **FOR:** *pathconf*() and *fpathconf*()
IF {_POSIX_PRIO_IO} is defined when <unistd.h> is included **THEN**
SETUP: Include the header <unistd.h>.
TEST: A call *function*() with a *name* parameter equal to {_POSIX_PRIO_IO} returns a value equal to {_POSIX_PRIO_IO}.
ELSE *NO_OPTION*
Conformance for pathconf, fpathconf: PASS, NO_OPTION

5 **FOR:** *pathconf*() and *fpathconf*()
SETUP: Include the header <unistd.h>.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: A call *function*() with a *name* parameter equal to {_POSIX_PRIO_IO} that refers to a file other than a directory returns a value corresponding to the option {_POSIX_PRIO_IO} for that file.

Conformance for pathconf, fpathconf: PASS

6 **FOR:** *pathconf*() and *fpathconf*()

IF {_POSIX_SYNCH_IO} is defined when <unistd.h> included **THEN**

SETUP: Include the header <unistd.h>.

TEST: A call *function*() with a *name* parameter equal to {_PC_SYNC_IO} returns a value equal to {_PC_SYNC_IO}.

ELSE *NO_OPTION*

Conformance for pathconf, fpathconf: PASS, NO_OPTION

7 **FOR:** *pathconf*() and *fpathconf*()

SETUP: Include the header <unistd.h>.

TEST: A call *function*() with a *name* parameter equal to {_PC_SYNC_IO} that refers to a file other than a directory returns a value corresponding to the option {_POSIX_SYNC_IO} for that file.

Conformance for pathconf, fpathconf: PASS

D_1 **IF** a PCD.lb documents the following **THEN**

TEST: A PCD.lb that documents whether an implementation supports an association of the variable name with the specified file, if the *path* argument to *pathconf*() or *fildes* argument to *pathconf*() refers to a directory, does so in 5.7.1.3.

ELSE *NO_OPTION*

Conformance for pathconf, fpathconf: PASS, NO_OPTION

5.7.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

5.7.1.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 6: Input and Output Primitives

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b{3} assertions.

6.1 Pipes

6.1.1 Create an Inter-Process Channel

Function: *pipe()*.

6.1.1.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b{3} assertions.

6.1.1.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.1.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b{3} assertions.

6.1.1.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.2 File Descriptor Manipulation

6.2.1 Duplicate an Open File Descriptor

Function: *dup(), dup2()*.

6.2.1.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b{3} assertions.

6.2.1.2 Description

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.2.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b{3} assertions.

6.2.1.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b{3} assertions.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.3 File Descriptor Deassignment

6.3.1 Close a File

Function: *close()*

6.3.1.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b{3} assertions

6.3.1.2 Description

1 **IF** {_POSIX_ASYNCHRONOUS_IO} **THEN**
TEST: An asynchronous I/O operation that is not canceled completes as if the *close()* operation had not yet occurred.
ELSE *NO_OPTION*
Conformance for close: PASS, NO_TEST, NO_OPTION

2 **IF** {_POSIX_ASYNCHRONOUS_IO}**THEN**
TEST: All operations that are not canceled complete as if the *close()* blocked until the operations completed
ELSE *NO_OPTION*
Conformance for close: PASS, NO_TEST, NO_OPTION

D-1 **TEST:** The PCD.1b documents whether any I/O operations are canceled, and which I/O operation may be canceled upon a call to the *close()* function, and does so in 6.3.1.2.
Conformance for close: PASS

3 **IF** *PCTS_shm_open* and *PCTS_ mmap* **THEN**
SETUP: Create and map a memory object that remains referenced at the last close.
TEST: After a call to *close(),* the entire contents of the memory object persist until the memory object becomes unreferenced.
TR: Test for shared memory.

If {POSIX_MAPPED_FILES}: test for memory mapped files.
ELSE *NO_OPTION*
Conformance for close: PASS, NO_OPTION

4 **IF** *PCTS_shm_open* and *PCTS_mmap* **THEN**
SETUP: Create a memory object and map it in at least two processes. Then unlink the memory object. Perform the last close of the memory object such that the close results in the memory object becoming unreferenced.
TEST: A call to *close()* removes the memory object.
ELSE *NO_OPTION*
Conformance for close: PASS, NO_OPTION

6.3.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause, there are no POSIX.1b {3} assertions.

6.3.1.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b{3} assertions.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4 Input and Output

6.4.1 Read from a File

Function: *read()*

6.4.1.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b{3} assertions

6.4.1.2 Description

1 **IF** *PCTS_read* and {POSIX_SYNCHRONIZED_IO) **THEN**
SETUP: Open a file by calling *open()* with O_RSYNC and O_DSYNC set in the *oflag* parameter.
TEST: A read operation initiated by calling *read()* either completes by transferring an image of the data to the requesting process or, if unsuccessful, by diagnosing and returning an indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityRead in 2.2.2.119.
Conformance for read: PASS, NO_TEST, NO_OPTION

2 **IF** *PCTS_read* and {POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by calling *open()* with O_RSYNC and O_DSYNC set in the *oflag* parameter.
TEST: When the synchronized read operation is initiated by calling *read()*, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data.
TR: Test for regular files.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWbeforeR in 2.2.2.119.
Conformance for read: PASS, NO_TEST, NO_OPTION

3 **IF** *PCTS_read* and {POSIX_SYNCHRONIZED_IO) **THEN**
SETUP: Open a file by calling *open()* with O_RSYNC and O_DSYNC set in the *oflag* parameter.
TEST: When the synchronized read operation is initiated by calling *read()*, any pending write request affecting the data to be read are written to the physical medium containing the file prior to reading the data, and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode
2. File serial number
3. ID of device containing this file
4. Number of links
5. User ID of the owner of the file
6. Group ID of the group of the file
7. The file size in bytes
8. Time of last access

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9. Time of last data modification

10. Time of last file status change.

TR: Test for regular files.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIOFileIntegrityRead in 2.2.2.120.
Conformance for read: PASS, NO_TEST, NO_OPTION

D-1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of a call to the *read()* function when *fildes* refers to a shared memory object does so in 6.4.1.2.
ELSE *NO_OPTION*
Conformance for read: PASS, NO_OPTION

6.4.1.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.4.1.4. Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.4.2 Write to a File

Function: *write()*.

6.4.2.1. Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.4.2.2. Description

1 **IF** *PCTS_write* and {POSIX_SYNCHRONIZED_IO) **THEN**
SETUP: Open a file by calling *open()* with O_DSYNC set in the *oflag* parameter.
TEST: A write operation initiated by calling *write()* either completes by transferring an image of the data to the physical medium containing the file or, if unsuccessful, by diagnosing and returning an indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWrite in 2.2.2.119.
Conformance for write: PASS, NO_TEST, NO_OPTION

2 **IF** *PCTS_write* and {POSIX_SYNCHRONIZED_IO) *PCTS_write* and {POSIX_SYNCHRONIZED_IO) **THEN**
SETUP: Open a file by calling *open()* with O_SYNC set in the *oflag* parameter.
TEST: At the time that the synchronized write operation initiated by calling *write()* occurs, the data are written to the physical medium containing the file and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode
2. File serial number
3. ID of the device containing this file
4. Number of links
5. User ID of the owner of the file

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6. Group ID of the group of the file
7. The file size in bytes
8. Time of last access
9. Time of last data modification
10. Time of last file status change.

TR: Test for regular files.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIOFileIntegrityWrite in 2.2.2.120.
Conformance for write: PASS, NO_TEST, NO_OPTION

D-1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of a call to the *write()* function when *fildes* refers to a shared memory object does so in 6.4.2.2.
ELSE *NO_OPTION*
Conformance for write: PASS, NO_OPTION

6.4.2.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.4.2.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.5 Control Operations on Files

6.5.1 Data Definitions for File Control Operations

1 **SETUP:** Include the header `<fcntl.h>`.
TEST: The constants O_DSYNC, O_RSYNC, and O_SYNC are defined and are unique values with respect to each other and to O_CREAT, O_EXCL, O_NOCTTY, O_TRUNC, O_APPEND, O_NONBLOCK, O_RDONLY, O_RDWR, AND O_WRONLY.
Conformance for fcntl.h: PASS

NOTE: IEEE Std 2003.1-1992 {4} was not as explicit as the above assertion regarding the uniqueness of the constants.

6.5.2 File Control

Function: *fcntl()*.

6.5.2.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.5.2.2 Description

All of the assertions for the Description clause of *fcntl()*, except for those relating to file locking and setting file status flags, were rewritten from IEEE Std 2003.1-1992 {4} because of the requirement that they apply to shared memory objects and the change in the form of assertions between IEEE Std 2003.1-1992 {4} and this standard.

(IEEE Std 2003.1-1992 {4})04
UNUSED

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

(IEEE Std 2003.1-1992 {4})05
UNUSED

(IEEE Std 2003.1-1992 {4})06
UNUSED

IEEE Std 2003.1-1992 {4})07
UNUSED

IEEE Std 2003.1-1992 {4})08
UNUSED

IEEE Std 2003.1-1992 {4})09
UNUSED

IEEE Std 2003.1-1992 {4})10
UNUSED

IEEE Std 2003.1-1992 {4})11
UNUSED

IEEE Std 2003.1-1992 {4})12
UNUSED

4 **TEST:** A call *fcntl(fildes, F_DUPFD, arg)* creates a new file descriptor that

1. Is the lowest numbered one available greater than or equal to the argument *arg*
2. Refers to the same open file description (file pointer, access mode, and file status flags) as the original file descriptor
3. Shares the same locks as the original file descriptor, if *fildes* does not refer to a shared memory object
4. Has the FD_CLOEXEC flag clear.

TR: Test for a regular file. If *PCTS_shm_open:* Test for *fildes* referring to a shared memory object.
Conformance for fcntl: PASS

5 **TEST:** *A* call *fcntl (fildes,F_GETFD)* returns the status of the file descriptor flags.
TR: Test for a regular file. If *PCTS_shm_open*: Test for *fildes* referring to a shared memory object.
Conformance for fcntl: PASS

6 **FOR:** *execl(), execv(), execle(), execve(), execlp()* and *execvp().*
TEST: A call *fcntl (fildes, F_SETFD, arg),* where the FD_CLOEXEC flag in *arg* is nonzero, sets the close-on-exec flag for the file associated with *filedes*and *function().*
TR: Test for a regular file. If *PCTS_shm_open:* Test for *fildes* referring to a shared memory object.
Conformance for fcntl: PASS

7 **FOR:** *execl(), execv(), execle(), execve(), execlp(),* and *execvp().*
TEST: A call *fcntl (fildes, F_SETFD, arg)*, where the FD_CLOEXEC flag in *arg* is nonzero, sets the close-on-exec flag for the file associated with *filedes*and *function().*
TR: Test for a regular file. If *PCTS_shm_open:* Test for *fildes* referring to a shared memory object.
Conformance for fcntl: PASS

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8 **TEST:** A call *fcntl(fildes, F_GETFL)* returns the file status flags O_APPEND, O_DSYNC, O_NONBLOCK, O_RSYNC, and O_SYNC, and the file access modes O_RDONLY, O_RDWR and O_WRONLY.
TR: Test for a regular file. If *PCTS_shm_open:* Test for *fildes* referring to a shared memory object.
Conformance for fcntl: PASS

9 **TEST:** A call *fcntl(fildes ,F_SETFL, 0)* sets the status flags for the file referenced by *fildes* to 0.
TR: Test for regular files.
Conformance for fcntl: PASS

10 **TEST:** A call *fcntl(fildes, F_SETFL, arg)* sets the status flags for the file referenced by *fildes* to values in *arg*.
TR: Test with *arg* having the values O_APPEND and O_NONBLOCK, individually and together, for regular files.
If {_POSIX_SYNCHRONIZED_IO}: Test with *arg* having the values O_APPEND, O_DSYNC, O_NONBLOCK, O_RSUNC, and O_SYNC, individually and all together, on a file supporting {_POSIX_SYNCHRONIZED_IO}.
Conformance for fcntl: PASS

D-1 **IF** *PCTS_shm_open* and PCD.1b that documents the following **THEN**
TEST: A PCD.1b that documents the effect of the values F_SETFL, F_GETLK, F_SETLK, and F_SETLKW for the argument *cmd* of the function *fcntl(),* when the file descriptor *fildes* refers to a shared member object, does so in 6.5.2.2.
ELSE *NO_OPTION*
Conformance for fcntl: PASS, NO_OPTION.

6.5.2.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.5.2.4 Errors

(IEEE Std 2003.1-1992 {4})41
UNUSED

(IEEE Std 2003.1-1992 {4})42
UNUSED

(IEEE Std 2003.1-1992 {4})48
UNUSED

(IEEE Std 2003.1-1992 {4})49
UNUSED

41 **TEST:** A call *fcntl(fildes, F_DUPFD,arg),* where *arg* is negative, returns -1 and sets *errno* to [EINVAL].
TR: Test for a regular file. If PCTS_*shm_open:* Test for *fildes* referring to a shared memory object.
Conformance for fcntl: PASS

42 **IF** a call *sysconf()* with an argument of {_SC_OPEN_MAX} does not return -1 **THEN**
TEST: A call *fcntl(fcldes, F_DUPFD, arg),* where *arg* is greater than or equal to [OPEN_MAX], returns -1 and sets *errno* to [EINVAL].
TR: Test for a regular file. If PCTS_*shm_open:* Test for *fildes* referring to a shared memory object.
ELSE *NO_TEST_SUPPORT*
Conformance for fcntl: PASS, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

42.1 **IF** a call *sysconf()* with an argument of {_SC_OPEN_MAX} returns -1 **THEN**
TEST: A call *fcntl(fcldes, F_DUPFD, arg),* where *arg* is *equal to* [PCTS_OPEN_MAX], returns a valid file descriptor and does not set *errno* to [EINVAL].
TR: Test for a regular file. If PCTS*_shm_open:* Test for *fildes* referring to a shared memory object.
ELSE *NO_TEST_SUPPORT*
Conformance for fcntl: PASS, NO_TEST_SUPPORT

42.2 **IF** *PCTS_NO_SYNC_IO_FILE* **THEN**
TEST: A call *fcntl(fcldes, F_SETL, arg*), where any of the file status flags O_DSYNC, O_RSYNC, OR O_SYNC are set in the *arg* argument, returns -1 and sets *errno* to [EINVAL].
TR: Test for a file that does not support synchronized I/O.
ELSE *NO_OPTION*
Conformance for fcntl: PASS, NO_OPTION

48 **IF** {OPEN_MAX}≤ *PCTS_OPEN_MAX* **THEN**
SETUP: Open {OPEN_MAX}*files.*
TEST: A call *fcntl(fildes, F_DUPD, arg)* returns -1 and sets *errno* to [EMFILE].
TR: Test for a regular file. If PCTS*_shm_open:* Test for *fildes* referring to a shared memory object.
ELSE *NO_TEST_SUPPORT*
Conformance for fcntl: PASS, NO_TEST_SUPPORT

48.1 **IF** {OPEN_MAX} > *PCTS_OPEN_MAX* **THEN**
SETUP: Open PCTS_OPEN_MAX files.
TEST: A call *fcntl(fildes, F_DUPD, arg)* returns a valid file descriptor and does not set *errno* to [EMFILE].
TR: Test for a regular file. If PCTS*_shm_open:* Test for *fildes* referring to a shared memory object.
ELSE *NO_TEST_SUPPORT*
Conformance for fcntl: PASS, NO_TEST_SUPPORT

49 **IF** {OPEN_MAX} ≤ *PCTS_OPEN_MAX* **THEN**
SETUP: Open {OPEN_MAX} files then make a file descriptor available whose value will be less than that specified in the *arg* support to *fcntl().*
TEST: A call *fcntl(fildes, F_DUPD, arg),* where *arg* is greater than the lowest available file descriptor, returns -1 and sets *errno* to [EMFILE].
TR: Test for a regular file. If PCTS*_shm_open:* Test for *fildes* referring to a shared memory object.
ELSE *NO_TEST_SUPPORT*
Conformance for fcntl: PASS, NO_TEST_SUPPORT

49.1 **IF** {OPEN_MAX} > *PCTS_OPEN_MAX* **THEN**
SETUP: Open *PCTS_OPEN_MAX* files then make a file descriptor available whose value will be less than that specified in the *arg* argument to *fcntl().*
TEST: A call *fcntl(fildes, F_DUPD, arg)* returns a valid file descriptor greater than or equal to the *arg* argument and does not set *errno* to [EMFILE].
TR: Test for a regular file. If PCTS*_shm_open:* Test for *fildes* referring to a shared memory object.
ELSE *NO_TEST_SUPPORT*
Conformance for fcntl: PASS, NO_TEST_SUPPORT

6.5.3 Reposition Read/Write File Offset

Function: *lseek().*

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.5.3.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.5.3.2 Description

D-1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of calling the *lseek()* function when *fildes* refers to a shared memory object does so in 6.5.3.2.
ELSE *NO_OPTION*
Conformance for lseek: PASS, NO_OPTION

6.5.3.3. Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.5.3.4. Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

6.6 File Synchronization

D-1 **TEST:** The PCD.1b documents the hardware characteristics, upon which the implementation relies to ensure that data is successfully transferred for synchronized I/O operation, and does so in in 6.6.
Conformance for syncio: PASS

6.6.1 Synchronize the State of a File

Function: *fsync()*.

6.6.1.1 Synopsis

1

M_GA_stdC_proto_decl(int;fsync; intfildes; unistd.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3
Conformance for fsync: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(fsync; unistd.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3
Conformance for fsync: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; fsync; unistd.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4
Conformance for fsync: PASS, NO_OPTION

4

M_GA_macro_args (fsync; unistd.h;;;)
SEE Assertion GA_macro_args in 2.7.3
Conformance for fsync: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.6.1.2 Description

D-1 **IF** PCTS_*fsync* **THEN**
TEST: The PCD.1b documents the manner in which the *fsync()* function transfers all data to the storage device associated with the open file description specified by the *fildes* argument in 6.6.1.2
ELSE *NO_OPTION*

5 **IF** PCTS_*fsync* **THEN**
SETUP: Write data to a file so that an error condition will not exist when *fsync()* is called.
TEST: The *fsync()* function does not return until the system has completed transferring all data to the storage device associated with the open file description specified by the *fildes* argument.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fsync: PASS, NO_TEST, NO_OPTION

R-1 **IF** PCTS_*fsync* **THEN**
TEST: A call to the *fsync()* function returns an error that is detected.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertions in 6.6.1.4

D-2 **IF** PCTS_*fsync* **THEN**
TEST: The PCD.1b documents sufficient information for the user to determine whether it is possible to configure an application and installation to ensure that the data is stored with the degree of required stability for the intended use of the *fsync()* function in 6.6.1.2.
ELSE *NO_OPTION*
Conformance for fsync: PASS, NO_OPTION

6 **IF** PCTS_*fsync* and {POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file for read-write access without specifying the O_DSYNC, O_RSYNC, or O_DSYNC in the *oflags* argument of *open()*. Then write some data to the file and read from another place in the file.
TEST: A call to the *fsync()* function forces all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O file integrity completion state; that is, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data. The following file attributes are also written to the physical medium containing the file prior to returning to the calling process:
1. File mode
2. File serial number
3. ID of device containing this file
4. Number of links
5. User ID of the owner of the file
6. Group ID of the group of the file
7. The file size in bytes
8. Time of last access
9. Time of last data modification
10. Time of last file status change.

NOTE: There is no known portable test method for this assertion.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ELSE *NO_OPTION*
Conformance for fsync: PASS, NO_TEST, NO_OPTION

7 **IF** PCTS_*fsync* and {POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file for read-write access without specifying the O_DSYNC, O_RSYNC, or O_DSYNC in the *oflags* argument of *open()*. Then write some data to the file.
TEST: A call to the *fsync()* function forces all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O file integrity completion state; that is the data are written to the physical medium containing the file, and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode
2. File serial number
3. ID of device containing this file
4. Number of links
5. User ID of the owner of the file
6. Group ID of the group of the file
7. The file size in bytes
8. Time of last access
9. Time of last data modification
10. Time of last file status change.

NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fsync: PASS, NO_TEST, NO_OPTION

6.6.1.3 Returns

R-2 **IF** PCTS_*fsync* **THEN**
TEST: Upon successful completion, the *fsync()* function returns 0.
ELSE *NO_OPTION*
SEE: Assertions in 6.6.1.2.

8 **TEST:** Upon unsuccessful completion, the *fsync()* returns -1 and set *errno* to indicate the error.
SEE: Assertions in 6.6.1.4.
Conformance for fsync: PASS

6.6.1.4 Errors

9 **IF** PCTS_*fsync* **THEN**
SETUP: Call the *fsync()* function with a *fildes* argument that is not a valid file descriptor.
TEST: *fsync()* returns -1 and set *errno* to [EBADF].
ELSE *NO_OPTION*
Conformance for fsync: PASS, NO_OPTION

10 **IF** PCTS_*fsync* **THEN**
IF PCTS_NO_SYNC_IO_FILE **THEN**
SETUP: Call the *fsync()* function with the *fildes* argument that is pointing to a file that does not support synchronized I/O.
TEST: *fsync()* returns -1 and set *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
Conformance for fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** not PCTS_*fsync* **THEN**
SETUP: Call the *fsync()* function, even though it is not supported by this implementation.
TEST: *fsync()* returns -1 and set *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for fsync: PASS, NO_OPTION

12 **IF** PCTS_*fsync* **THEN**
TEST: In the event that any of the queued I/O operations fail, *fsync()* returns the error conditions defined for *read()* and *write()*.
ELSE *NO_OPTION*
Conformance for fsync: PASS, NO_TEST, NO_OPTION

6.6.2 Synchronize the Data of a File

Function: *fdatasync*().

6.6.2.1 Synopsis

1
M_GA_stdC_Proto_decl(int; fdatasync: int fildes; unistd.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3
Conformance for fdatasync: PASS[1, 2] NO_OPTION

2
M_GA_commonC_int_result_decl(int; fdatasync: int fildes; unistd.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3
Conformance for fdatasync: PASS[1, 2] NO_OPTION

3
M_GA_macro_result_decl(int; fdatasync: int fildes; unistd.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4
Conformance for fdatasync: PASS[1, 2] NO_OPTION

4
M_GA_macro_args(int; fdatasync: int fildes; unistd.h;;;)
SEE: Assertion GA_macro_result_decl in 2.7.3
Conformance for fdatasync: PASS[1, 2] NO_OPTION

6.6.2.2 Description

5 **IF** PCTS_*fdatasync* **THEN**
SETUP: Write data to a file so that an error condition will not exist when *fdatasync()* is called.
TEST: The *fdatasync()* function does not return until the system has completed transferring all data to the storage device associated with the open file description specified by the *fildes* argument.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_TEST, NO_OPTION

R_1 **IF** PCTS_*fdatasync* **THEN**
TEST: A call to the *fdatasync()* function returns an error that is detected.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertions in 6.6.2.4

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6 **IF** PCTS_*fdatasync* **THEN**
SETUP: Open a file for read-write access without specifying O_DSYNC, O_RSYNC, or O_DSYNC in the *oflags* argument of *open()*. Then write some data to the file and read from another place in the file.
TEST: A call to the *fdatasync()* function forces all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O data integrity completion state; that is, it completes by transferring an image of the data to the requesting process, and any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_TEST, NO_OPTION

7 **IF** PCTS_*fdatasync* **THEN**
SETUP: Open a file for read-write access without specifying O_DSYNC, O_RSYNC, or O_DSYNC in the *oflags* argument of *open()*. Then write some data to the file.
TEST: A call to the *fdatasync()* function forces all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O data integrity completion state; that is the data are written to the physical medium containing the file.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_TEST, NO_OPTION

6.6.2.3 Returns

R_2 **IF** PCTS_*fdatasync* **THEN**
TEST: Upon successful completion, the *fdatasync()* function returns 0.
ELSE *NO_OPTION*
SEE: Assertions in 6.6.2.2.

8 **TEST:** Upon unsuccessful completion, the *fdatasync()* returns -1 and sets *errno* to indicate the error.
SEE: Assertions in 6.6.2.4.
Conformance for fdatasync: PASS

6.6.2.4 Errors

9 **IF** PCTS_*fdatasync* **THEN**
SETUP: Call the *fdatasync()* function with a *fildes* argument that is not a valid file descriptor.
TEST: *fdatasync*() returns -1 and sets *errno* to [EBADF].
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_OPTION

10 **IF** PCTS_*fdatasync* **THEN**
IF PCTS_NO_SYNC_IO_FILE **THEN**
SETUP: Call the *fdatasync()* function with the *fildes* argument pointing to a file that does not support synchronized I/O.
TEST: *fdatasync*() returns -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_OPTION

11 **IF** not PCTS_*fdatasync* **THEN**
SETUP: Call the *fdatasync()* function even though it is not supported by this implementation.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: *fdatasync*() returns -1 and sets *errno* to [ENOSYS].
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_OPTION

12 **IF** PCTS_*fdatasync* **THEN**
TEST: In the event that any of the queued I/O operations fail, *fdatasync()* returns the error conditions defined for *read()* and *write()*.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_TEST, NO_OPTION

6.7 Asynchronous Input and Output

6.7.1 Data Definitions for Asynchronous Input and Output

There are no requirements for conforming implementations in this clause.

6.7.1.1 Asynchronous I/O Control Block

1 **SETUP:** Include the header `<aio.h>`.
TEST: An asynchronous I/O control block structure *aiocb* is defined and has at least the following members with the indicated types:

Member Type	Member Name	Description
int	*aio_fildes*	File descriptor
off_t	*aio_offset*	File offset
*volatile void**	*aio_buf*	Location of buffer
size_t	*aio_nbytes*	Length of transfer
int	*aio_reqprio*	Request priority offset
struct sigevent	*aio_sigevent*	Signal number and value
int	*aio_lio_opcode*	Operation to be performed

Conformance for aio.h: PASS

D-1 **TEST:** The PCD.1b documents any added extensions [as permitted in 1.3.1.1 item (2)] made to the *aiocb* asynchronous I/O control block structure, including the definition of an environment in which an application can be run with the behavior specified byPOSIX.1b {3}, in 6.7.1.1
Conformance for aio.h: PASS

2 **TEST:** Added extensions to the *aiocb* structure, are enabled as required by 1.3.1.1 that is, a Strictly Conforming Application does not need to be modified to execute in such a modified environment.
NOTE: There is no known portable test method for this assertion.
Conformance for aio.h: PASS, NO_TEST

3 **FOR:** *aio_read(), aio_write(),* and *aio_lio_listio()*
IF *PCTS_function* **THEN**
SETUP: Open a file so that O_APPEND is not set for the file descriptor *aio_fildes,* and *aio_fildes* is associated with a device that is capable of seeking.
TEST: The operation requested by *function()* takes place at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to the operation with an *offset* argument equal to *aio_offset* and a *whence* argument equal to SEEK_SET.
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for aio.h: PASS, NO_OPTION

4 **FOR:** *aio_write(),* and *aio_lio_listio()*
IF *PCTS_function* **THEN**
IF *PCTS_APPEND_WRITE_SAME_ORDER* **THEN**
SETUP: Open a file so that O_APPEND is set for the file descriptor. Open another file where *aio_fildes* is associated with a device that is incapable of seeking.
TEST: Write operations for the *function()* function append to the file in the same order as the calls were made.
TR: Test for both files in the SETUP. Be sure to use the same priority for all writes.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **TEST:** The PCD.1b documents all circumstances that relax the restriction to have write append operations occur in the same order as the write calls, in 6.7.1.1.
Conformance for aio.h: PASS

5 **FOR:** *aio_read(), aio_write(),* and *aio_lio_listio()*
IF PCTS_*function* and {_POSIX_PRIORITIZED_IO}
and {_POSIX_PRIORITY_SCHEDULING} **THEN**
TEST: Asynchronous I/O is queued in priority order, with the priority of each asynchronous operation based on the current scheduling priority of the calling process.
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_OPTION

6 **FOR:** *aio_read(), aio_write(),* and *aio_lio_listio()*
IF *PCTS_function* and {_POSIX_PRIORITIZED_IO}
and {_POSIX_PRIORITY_SCHEDULING} **THEN**
TEST: The *aio_reqprio* member can be used to lower, but not raise, the asynchronous I/O operation priority when it is within the range zero through {AIO_PRIO_DELTA_MAX}, inclusive.
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_OPTION

7 **FOR:** *aio_read(), aio_write()* and *aio_lio_listio()*
IF *PCTS_function* and {POSIX_PRIORITY_SCHEDULING} **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: The priority of an asynchronous request is computed as process scheduling priority minus *aio_reqprio.*
TR: Test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **FOR:** *aio_read() aio_write,* and *aio_lio_listio()*
IF *PCTS_function* and {POSIX_PRIORITIZED_IO} **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: Requests issued by *function(),* with the same priority, to a character special file are processed by the underlying device in FIFO order.
TR: Test for both files in the SETUP. Be sure to use the same priority for all writes.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_TEST_SUPPORT, NO_OPTION

D_3 **IF** a PCD.1b documents the following **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: A PCD.1b that documents the order of processing of asynchronous I/O requests of the same priority issued to files that are not character special files does so in 6.7.1.1.

ELSE *NO_OPTION*

Conformance for aio.h: PASS, NO_OPTION

9 **FOR:** *aio_read() aio_write,* and *aio_lio_listio()*

IF *PCTS_function* and {POSIX_PRIORITIZED_IO} and {POSIX_PRIORITY_SCHEDULING} **THEN**

TEST: The value of *aio_reqprio* has no effect on process scheduling priority.

ELSE *NO_OPTION*

Conformance for aio.h: PASS, NO_OPTION

10 **FOR:** *aio_read() aio_write,* and *aio_lio_listio()*

IF *PCTS_function* and {POSIX_PRIORITIZED_IO} and {POSIX_PRIORITY_SCHEDULING} **THEN**

SETUP: Create prioritized asynchronous I/O requests to the same file that are blocked waiting for a resource required for that I/O operation.

TEST: The higher-priority asynchronous I/O requests are granted the resource before lower-priority I/O requests.

ELSE *NO_OPTION*

Conformance for aio.h: PASS, NO_OPTION

D_4 **TEST:** The PCD.1b documents the relative priority of asynchronous and synchronous I/O requests in 6.7.1.1.

Conformance for aio.h: PASS

11 **IF** {_POSIX_PRIORITIZED_IO} **THEN**

TEST: The PCD.1b documents for which files I/O prioritization is supported in 6.7.1.1.

ELSE *NO_OPTION*

Conformance for aio.h: PASS, NO_OPTION

12 **FOR:** *aio_read*(), *aio_write*(), and *aio_lio_listio*()

IF *PCTS_function* **THEN**

SETUP: Use an *aiocb* where the *aio_sigevent.sigev_notify* member is SIGEV_NONE

TEST: After a call to the *function*() function, no signal is posted upon I/O completion, and the error status for the operation and the return status for the operation are set appropriately.

ELSE *NO_OPTION*

Conformance for aio.h: PASS, NO_OPTION

13 **FOR:** *aio_read*(), *aio_write*(), and *aio_lio_listio*()

IF *PCTS_function* **THEN**

SETUP: Use an *aiocb* where the *aio_sigevent.sigev_notify* member is SIGEV_SIGNAL.

TEST: After a call to the *function*() function and upon I/O completion, the signal specified in *aio_sigevent.sigev_signo* is sent to the process.

ELSE *NO_OPTION*

Conformance for aio.h: PASS, NO_OPTION

14 **FOR:** *aio_read*(), *aio_write*(), and *aio_lio_listio*()

IF *PCTS_function* {_POSIX_REALTIME_SIGNALS} **THEN**

SETUP: Use an *aiocb* where the *aio_sigevent.sigev_notify* member is SIGEV_SIGNAL. Set the signal number to be queued in the range from SIGRTMIN to SIGRTMAX and then set the SA_SIGINFO flag for that signal number.

TEST: After a call to the *function*() and upon I/O completion, the signal is queued to the process, and the value specified in *aio_sigevent.sigev_value* is the *si_value* component of the generated signal.

ELSE *NO_OPTION*

Conformance for aio.h: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE: The following requirements of POSIX.1b {3} are explicitly required by the relevant functions and, therefore, are not considered General Assertions and are to be tested in the functions with the following requirements:

The return status of the asynchronous operation is the number of bytes transferred by the I/O operation. If the error status is set to indicate an error completion, then the return status is set to the return value that the corresponding *read*(), *write*(), or *fsynch*() call would have returned. When the error status is not equal to [EINPROGRESS], the return status shall reflect the return status of the corresponding synchronous operation.

6.7.1.2 Manifest Constants

15 **SETUP:** Include the header `<aio.h>`.
TEST: The symbols AIO_CANCELED, AIO_NOTCANCELED, and AIO_ALLDONE are defined and have unique values relative to each other.
Conformance for aio.h: PASS

16 **SETUP:** Include the header `<aio.h>`.
TEST: The symbols LIO_WAIT and LIO_NOWAIT are defined and have unique values relative to each other.
Conformance for aio.h: PASS

17 **SETUP:** Include the header `<aio.h>`.
TEST: The symbols LIO_READ, LIO_WRITE, and LIO_NOP are defined and have unique values relative to each other.
Conformance for aio.h: PASS

6.7.2 Asynchronous Read

Function; *aio_read*().

6.7.2.1 Synopsis

1
M_GA_stdC_proto_decl(int; aio_read; struct aiocb *aiocbp: aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3
Conformance for aio_read: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(aio_read; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3
Conformance for aio_read: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; aio_read; aio.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4
Conformance for aio_read: PASS, NO_OPTION

4
M_GA_macro_args(aio_read; aio.h;;;)
SEE: Assertion GA_macro_args in 2.7.3
Conformance for aio_read: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.7.6.2 Description

5 **IF** *PCTS_aio_read* **THEN**
TEST: A call to the *aio_read*() function reads *aiocbp->aio_nbytes* from the file associated with *aiocbp->aiofildes* into the buffer pointed to by *aiocbp->aio_buf.*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_OPTION

6 **IF** *PCTS_aio_read* **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: A call to the *aio_read*() function returns when the read request has been initiated or queued to the file or device, even when the data cannot be delivered immediately.
TR: Test for a character special device.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_aio_read* and {_POSIX_PRIORITIZED_IO} **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
SETUP: Use a file for which prioritized I/O is supported.
TEST: The asynchronous operation caused by the *aio_read*() function is submitted at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio.*
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_OPTION

8 **IF** *PCTS_aio_read* and {_POSIX_PRIORITIZED_IO} **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
SETUP: Use a file for which prioritized I/O is supported.
TEST: The asynchronous operation caused by the *aio_read*() function is submitted at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio.*
TR: Test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_aio_read* and *PCTS_aio_error* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_read*() function, when used as an argument to *aio_error*(), returns the error status of the asynchronous operation while it is proceeding.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST, NO_OPTION

10 **IF** *PCTS_aio_read* and *PCTS_aio_error* **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_read*() function, when used as an argument to *aio_error*(), returns the error status of the asynchronous operation while it is proceeding.
TR: Test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST SUPPORT, NO_OPTION

11 **IF** *PCTS_aio_read* and *PCTS_aio_return* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_read*() function, when used as an argument to *aio_return*(), returns the return status of the asynchronous operation while it is proceeding.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST, NO_OPTION

12 **IF** *PCTS_aio_read* and *PCTS_aio_return* **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_read*() function, when used as an argument to *aio_return*(), returns the return status of the asynchronous operation while it is proceeding.
TR: Test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST SUPPORT, NO_OPTION

R_1 **IF** *PCTS_aio_read* **THEN**
TEST: A call *aio_read*() returns an error condition encountered during queuing, without having initiated or queued the request.
ELSE *NO_OPTION*
SEE: Assertions in 6.7.2.4

13 **IF** *PCTS_aio_read* **THEN**
TEST: A call *aio_read*() causes the requested operation to take place at the absolute position in the file as given by *aio_offset*, as if *lseek*() were called immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to SEEK_SET.
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_OPTION

D_1 **IF** *PCTS_aio_read* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the value of the file offset for the file after a successful call to enqueue an asynchronous I/O operation, does so in 6.7.2.2.
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_OPTION

D_2 **IF** *PCTS_aio_read* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of a call to the *aio_read*() when the buffer pointed to by *aiocbpl->aio_buf* or the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, does so in 6.7.2.2.
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_OPTION

D_3 **IF** *PCTS_aio_read* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the results of simultaneous asynchronous operations using the same *aiocbp,* does so in 6.7.2.2.
ELSE NO_OPTION
Conformance for aio_read: PASS, NO_OPTION

15 **IF** *PCTS_aio_read* and {_POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by calling *open*() with O_RSYNC and O_DSYNC set in the *oflag* parameter.
TEST: A read operation initiated by calling *aio_read*() either completes by transferring an image of the data to the requesting process or, if unsuccessful, by diagnosing and returning and indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityRead in 2.2.2.119.
Conformance for aio_read: PASS, NO_TEST, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16 **IF** *PCTS_aio_read* and {_POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by calling *open*() with O_RSYNC and O_SYNC set in the *oflag* parameter.
TEST: At the time that the synchronized read operation initiated by calling *aio_read*() occurs, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data.
TR: Test for regular files.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWbeforeR in 2.2.2.119.
Conformance for aio_read: PASS, NO_TEST, NO_OPTION

17 **IF** *PCTS_aio_read* and {_POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by calling *open*() with O_RSYNC and O_SYNC set in the *oflag* parameter.
TEST: At the time that the synchronized read operation initiated by calling *aio_read*() occurs, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data, and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode
2. File serial number
3. ID of device containing this file
4. Number of links
5. User ID of the owner of the file
6. Group ID of the group of the file
7. The file size in bytes
8. Time of last access
9. Time of last data modification
10. Time of last file status change.

TR: Test for regular files.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIOFileIntegrityRead in 2.2.2.120.
Conformance for aio_read: PASS, NO_TEST, NO_OPTION

D_4 **IF** *PCTS_aio_read* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of any system action that changes the process memory space, while an asynchronous I/O is outstanding to the address range being changed, does so in 6.7.2.2.
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_OPTION

6.7.2.3 Returns

R_2 **IF** *PCTS_aio read* **THEN**
TEST: The *aio_read*() function returns the value zero to the calling process after the I/O operation is successfully queued.
ELSE *NO_OPTION*
SEE: Assertions in 6.7.2.2.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

R_3 **IF** *PCTS_aio read* **THEN**
TEST: The *aio_read*() function returns the value -1 to the calling process and sets *errno* to indicate the error when the I/O operation is not successfully queued.
ELSE *NO_OPTION*
SEE: Assertions in 6.7.2.4.

6.7.2.4 Errors

18 **IF** *PCTS_aio read* **THEN**
IF {AIO_MAX}≤ *PCTS_AIO_MAX* **THEN**
SETUP: Queue {AIO_MAX} asynchronous I/O operations before calling *aio_read*().
TEST: A call to the *aio_write*() function returns -1 and sets *errno* to [EAGAIN].
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_aio read* **THEN**
IF {AIO_MAX}≤ *PCTS_AIO_MAX* and *PCTS_GTI_DEVICE* **THEN**
SETUP: Queue {AIO_MAX} asynchronous I/O operations before calling *aio_read*().
TEST: A call to the *aio_read*() function returns -1 and sets *errno* to [EAGAIN].
TR: Test for a character special device.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

20 **IF** *PCTS_aio read* **THEN**
IF {AIO_MAX} *PCTS_AIO_MAX* and *PCTS_GIT_DEVICE* **THEN**
SETUP: Queue *PCTS_AIO_MAX* -1 asynchronous I/O operations before calling *aio_read*().
TEST: A call to the *aio_read*() function returns 0 and does not set *errno* to [EAGAIN].
TR: Test for a character special device.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **IF** not *PCTS_aio read* **THEN**
TEST: A call to the *aio_read*() function returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for aio_read: PASS, N0_OPTION

ebadf1 **IF** *PCTS_aio read* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *aio_read*() function, where the *aiocbp->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously, then the *aio_read*() function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

ebadf2 **IF** *PCTS_aio read* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *aio_read*() function, where the *aiocbp->aio_fildes* argument is not open for reading, the implementation either detects the

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

condition synchronously, then the *aio_read*() function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, then the return status for the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

einval1 **IF** *PCTS_aio read* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *aio_read*() function, where the file offset value implied by *aiocbp->aio_offset* would be invalid, or *aiocbp->aio_reqprio* is not a valid value, or *aiocbp->aio_nbytes* is an invalid value, the implementation either detects the condition synchronously, then the *aio_read*() function returns -1 and sets *errno* to [EINVAL]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set -1, and the error status of the asynchronous operation is set to [EINVAL].

TR: Test separately for each of the three conditions above.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

22 **IF** *PCTS_aio read* **THEN**

IF *PCTS_aio_cancel* and *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Call the *aio_read*() function so that it successfully queues the I/O operation.

TEST: After a read operation is canceled by a call to the *aio_cancel*() function, the return status of the asynchronous operation returned by a call to the *aio_return*() function is -1, and the error status returned by a call to the *aio_error*() function is [EINTR] or [ECANCELED].

TR: Test for a pipe and a FIFO.
If *PCTS_GTI_DEVICE*, test for a character special file.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

R_4 **IF** *PCTS_aio read* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *aio_read*() function, where the *aiocbp->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously, then the *aio_read*() function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1, and the value of the call to *aio_error*() is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion ebadf1 in 6.7.2.4.

R_5 **IF** *PCTS_aio read* **THEN**

IF *PCTS_aio_error* **THEN**

TEST: After a call to the *aio_read*() function, where the *aiocbp->aio_fildes* argument is not open for reading, the implementation either detects the condition asynchronously, then the *aio_read*() function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SEE: Assertion ebadf1 in 6.7.2.4.

23 **IF** *PCTS_aio read* and *PCTS_aio_cancel* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
SETUP: Call the *aio_read*() function so that it successfully queues the I/O operation.
TEST: A read operation, that is canceled after a call to the *aio_cancel*() function and before the I/O is completed, results in a return status returned by a call to the *aio_return*() function that is -1 and an error status returned by a call to the *aio_error*() function that is [ECANCELED].
TR: Test for both a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

R_6 **IF** *PCTS_aio read* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *aio_read*() function, where the file offset value implied by *aiocbp->aio_offset* would be invalid, the implementation detects the condition asynchronously, then the return status of the asynchronous operation is set to -1and the value of the call to *aio_error*() is set to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion einval in 6.7.2.4.

6.7.3 Asynchronous Write

Function: *aio_write*().

6.7.3.1 Synopsis

1
M_GA_stdC_proto_decl(int; aio_write; struct aiocb * aiocbp; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3
Conformance for aio_write: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(aio_write; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3
Conformance for aio_write: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; aio_write; aio.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4
Conformance for aio_write: PASS[1, 2], NO_OPTION

4
M_GA_macro_args (aio_write; aio.h;;;)
SEE: Assertion GA_macro_args in 2.7.3
Conformance for aio_write: PASS, NO_OPTION

6.7.3.2 Description

5 **IF** *PCTS_aio_write* **THEN**
TEST: A call to the *aio_write*() function writes *aiocbp->aio_nbytes* to the file associated with *aiocbp->aio_fildes* from the buffer pointed to by *aiocbp->aio_buf*.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

6 **IF** *PCTS_aio_write* **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: A call to the *aio_write*() function returns when the write request has been initiated or queued to the file or device.
TR: Test for a character special device.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS,NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_aio_write* and {_POSIX_PRIORITIZED_IO} **THEN**
SETUP: Use a file for which prioritized I/O is supported.
TEST: The asynchronous operation caused by the *aio_write*() function is submitted at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio.*
NOTE: There is no known portable test method for this assertion.

ELSE *NO_OPTION*
Conformance for aio_write: PASS,NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_aio_write* and {_POSIX_PRIORITIZED_IO} **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
SETUP: Use a file for which prioritized I/O is supported.
TEST: The asynchronous operation caused by the *aio_write*() function is submitted at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio.*
TR: Test for a character special file.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS,NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_aio_write* and *PCTS_aio_error* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_write*() function, when used as an argument to *aio_error*(), returns the error status of the asynchronous operation while it is proceeding.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST, NO_OPTION

10 **IF** *PCTS_aio_write* and *PCTS_aio_error* **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_write*() function, when used as an argument to *aio_error*(), returns the error status of the asynchronous operation while it is proceeding.
TR: Test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_aio_write* and *PCTS_aio_return* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_write*() function, when used as an argument to *aio_return*(), returns the error status of the asynchronous operation while it is proceeding.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12 **IF** *PCTS_aio_write* and *PCTS_aio_return* **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_write*() function, when used as an argument to *aio_return*(), returns the return status of the asynchronous operation while it is proceeding.
TR: Test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_aio_write* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of a call to the *aio_write*(), when the buffer pointed to by *aiocbp->aio_buf* or the control block pointed to by *aiocbp*, becomes an illegal process prior to asynchronous I/O completion, does so in 6.7.3.2.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

13 **IF** *PCTS_aio_write* **THEN**
SETUP: Use a file where the O_APPEND flag is not set for the file descriptor *aio_fildes*.
TEST: A call *aio_write*() causes the requested operation to take place at the absolute position in the file as given by *aio_offset*, as if *lseek*() were called immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to SEEK_SET.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

14 **IF** *PCTS_aio_write* **THEN**
SETUP: Use a file where the O_APPEND flag is set for the file descriptor.
TEST: Write operations caused by calling the *aio_write*() function apend to the file in the same order as the calls were made.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

D_2 **IF** *PCTS_aio_write* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the value of the file offset for the file after a successful call to enqueue an asynchronous I/O operation does so in 6.7.3.2.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_aio_write* **THEN**
TEST: The *aiocbp->aio_lio_opcode* field is ignored by *aio_write*().
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

D_3 **IF** *PCTS_aio_write* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the results of simultaneous asynchronous operations using the same *aiocbp* does so in 6.7.3.2.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

16 **IF** *PCTS_aio_write* and {_POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by callilng *open*() with O_DSYNC set in the *oflag* parameter.
TEST: A write operation initiated by calling *aio_write*() either completes by transferring an image of the data to the physical medium containing the file or, if unsuccessful, by diagnosing and returning an indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWrite in 2.2.2.119
Conformance for aio_write: PASS, NO_TEST, NO_OPTION

17 **IF** *PCTS_aio_write* and {_POSIX_SYNCHRONIZED_IO} **THEN** *PCTS_aio_write* and {_POSIX_SYNCHRONIZED_IO}
SETUP: Open a file by calling *open*() with O_SYNC set in the *oflag* parameter.
TEST: When a synchronized write operation is initiated by calling *aio_write*(), the data are written to the physical medium containing the file, and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode
2. File serial number
3. ID of device containing this file
4. Number of links
5. User ID of the owner of the file
6. Group ID of the group of the file
7. The file size in bytes
8. Time of last access
9. Time of last data modification
10. Time of last file status change.

TR: Test for regular files.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIOFileIntegrityWrite in 2.2.2.120
Conformance for aio_write: PASS, NO_TEST, NO_OPTION

D_4 **IF** *PCTS_aio_write* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of any system action that changes the process memory space, while an asynchronous I/O is outstanding to the address range being changed, does so in 6.7.3.2.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

6.7.3.3 Returns

R_1 **IF** *PCTS_aio_write* **THEN**
TEST: The *aio_write*() function returns the value zero to the calling process after the I/O operation is successfully queued.
ELSE *NO_OPTION*
SEE: Assertions in 6.7.3.2.

R_2 **IF** *PCTS_aio_write* **THEN**
TEST: The *aio_write*() function returns the value -1 to the calling process and sets *errno* to indicate the error when the I/O operation is not successfully queued.
ELSE *NO_OPTION*
SEE: Assertions in 6.7.3.4.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.7.3.4 Errors

18 **IF** *PCTS_aio_write* **THEN**
IF {AIO_MAX} ≤ *PCTS_AIO_MAX* **THEN**
SETUP: Queue {AIO_MAX} asynchronous I/O operations before calling *aio_write*().
TEST: A call to the *aio_write*() function returns -1 and sets *errno* to [EAGAIN].
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_aio_write* **THEN**
IF {AIO_MAX} ≤ *PCTS_AIO_MAX* and **THEN**
SETUP: Queue {AIO_MAX} asynchronous I/O operations before calling *aio_write*().
TEST: A call to the *aio_write* function returns -1 and sets *errno* to [EAGAIN]
TR: Test for a character special device.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

20 **IF** *PCTS_aio_write* **THEN**
IF {AIO_MAX} *PCTS_AIO_MAX* and *PCTS_GTI_DEVICE* **THEN**
SETUP: Queue *PCTS_AIO_MAX* asynchronous I/O operations before calling *aio_write*().
TEST: A call to the *aio_write*() function returns 0 and does not set *errno* to [EAGAIN].
TR: Test for a character special device.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **IF** not *PCTS_aio_write* **THEN**
TEST: A call to the *aio_write*() function returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

ebadf1 **IF** *PCTS_aio_write* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* THEN
TEST: After a call to the *aio_write*() function, where the *aiocbp->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously, then the *aio_write*() function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

ebadf2 **IF** *PCTS_aio_write* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* THEN
TEST: After a call to the *aio_write*() function, where the *aiocbp->aio_fildes* argument is not open for writing, the implementation either detects the condition synchronously, then the *aio_write*() function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

einval1 **IF** *PCTS_aio_write* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* THEN
TEST: After a call to the *aio_write*() function, where the file offset value implied by *aiocbp->aio_offset* would be invalid, or *aiocbp->aio_reqprio* is not a valid value, or *aiocbp->aio_nbytes* is an invalid value, the implementation either detects the condition synchronously, then the *aio_write*() function returns -1 and sets *errno* to [EINVAL]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EINVAL]
TR: Test separately for each of the three conditions above.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

22 **IF** *PCTS_aio_write* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* THEN
SETUP: Call the *aio_write*() function so that it successfully queues the I/O operation for a pipe or FIFO that has the O_NONBLOCK flag set for the file descriptor and with the *aiocbp->aio_nbytes* less than or equal to {PIPE_BUF} and where there is insufficient capacity to accept the data.
TEST: The return status of the asynchronous operation returned by a call to the *aio_return*() function is -1 and the error status returned by a call to the *aio_error*() function is [EAGAIN].
TR: Test for both a pipe and FIFO. If {PIPE_BUF} is greater than *PCTS_PIPE_BUF*, test with values of *aiocbp->aio_nbytes* up to and including *PCTS_PIPE_BUF*.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

23 **IF** *PCTS_aio_write* **THEN**
IF *PCTS_aio_cancel* and *PCTS_aio_error* and *PCTS_aio_return* THEN
SETUP: Call the *aio_write*() function so that it successfully queues the I/O operation.
TEST: After a write operation is canceled by a call to the *aio_cancel*() function and no data were transferred, the return status of the asynchronous operation returned by a call to the *aio_return*() function is -1 and the error status returned by a call to the *aio_error*() function is [EINTR] or [ECANCELED].
TR: Test for both a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

24 **IF** *PCTS_aio_write* **THEN**
IF the implementation supports a maximum file size **THEN**
TEST: An attempt to write a file that would exceed an implementation-defined maximum size by calling the *aio_write*() function and where the write operation is successfully queued, causes the error status for the asynchronous operation to be [EFBIG].
NOTE: The assertion test would require an unreasonable amount of time or resources on most implementations.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

25 **IF** *PCTS_aio_write* and {_POSIX_JOB_CONTROL} **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SETUP: Make the testing process part of the background process group; set TOSTOP; ignore or block SIGTTOU signals; and make the process group of the process orphaned. Call the *aio_write*() function so that it successfully queues the I/O operation to write from its controlling terminal.

TEST: The write operation will fail; the return status of the asynchronous operation returned by a call to the *aio_return*() function is -1 and the error status returned by a call to the *aio_error*() function is [EIO].

TR: Test for both a pipe and a FIFO.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

26 **IF** *PCTS_aio_write* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* and *PCTS_DETECT_ENOSPC* **THEN**

SETUP: Fill a device so that there is no more space available on it for data. Call the *aio_write*() function so that it successfully queues the I/O operation.

TEST: After a call to the *aio_write*() function, the implementation either detects the condition synchronously, the *aio_write* () function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

27 **IF** *PCTS_aio_write* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* THEN

SETUP: Call the *aio_write*() function so that it successfully queues the I/O operation.

TEST: After a call to the *aio_write*() function to write to a pipe or FIFO that is not open for reading for any process, the implementation either detects the condition synchronously, the *aio_write*() function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

R_3 **IF** *PCTS_aio_write* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *aio_write*() function, where the *aiocbp_>aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition asynchronously, the *aio_write*() function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, the error status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion ebadf1 in 6.7.3.4.

R_4 **IF** *PCTS_aio_write* **THEN**

IF *PCTS_aio_error* **THEN**

TEST: After a call to the *aio_write*() function, where the *aiocbp_>aio_fildes* argument is not open for writing, the implementation either detects the condition synchronously, then the *aio_write*() function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
SEE: Assertion ebadf1 in 6.7.3.4.

28 **IF** *PCTS_aio_write* and *PCTS_aio_cancel* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
SETUP: Call the *aio_write*() function so that it successfully queues the I/O operation.
TEST: A write operation, that is canceled after call to the *aio_cancel*() function and before the I/O is completed, results in a return status returned by a call to the *aio_return*() function of -1 and the error status returned by a call to the *aio_error*() function of [ECANCELED].
TR: Test for both a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

R_5 **IF** *PCTS_aio_write* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *aio_write*() function, where file offset value implied by *aiocbp->aio_offset* would be invalid, the implementation detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion einval in 6.7.3.4.

6.7.4 List Directed I/O

Function *lio_listio*().

6.7.4.1 Synopsis

1

M_GA_stdC_proto_decl(int; lio_listio; int mode, struct aiocb *const list[], int nent, struct sigevent *sig; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for lio_listio: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(lio_listio; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for lio_listio: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; lio_listio; aio.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for lio_listio: PASS, NO_OPTION

4

M_GA_macro_args (lio_listio; aio.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for lio_listio: PASS, NO_OPTION

6.7.4.2 Description

5 **IF** *PCTS_lio_listio* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: A successful call to the *lio_listio*() function, where the *mode* argument is LIO_WAIT, waits until all I/O is complete, ignores the *sig* argument, and returns 0.

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

6 **IF** *PCTS_lio_listio* **THEN**

TEST: A successful call to the *lio_listio*() function, where the *mode* argument is LIO_NOWAIT and the *sig* argument is **NULL**, returns immediately, does not deliver a signal upon completion of all I/O operations, and returns 0.

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

7 **IF** *PCTS_lio_listio* **THEN**

TEST: A successful call to the *lio_listio*() function, where the *mode* argument is LIO_NOWAIT, the *sig* argument is not **NULL**, and the *sigev_signo* member of the *sigevent* structure referenced by *sig* is zero, returns immediately, does not deliver a signal upon completion of all I/O operations, and returns 0.

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

8 **IF** *PCTS_lio_listio* **THEN**

TEST: A successful call to the *lio_listio*() function, where the *mode* argument is LIO_NOWAIT , the *sig* argument is not **NULL**, and the *sigev_signo* member of the *sigevent* structure referenced by *sig* is not zero, then the signal number indicated by *sigev_signo* is delivered when all the requests in *list* have completed and the call immediately returns a value of zero.

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

D_1 **IF** *PCTS_lio_listio* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the order in which the I/O requests enumerated by *list* are submitted does so in 6.7.4.2.

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

9 **IF** *PCTS_lio_listio* **THEN**

TEST: A successful call to the *lio_listio*() function, with **NULL** elements in the *list* argument, ignores the **NULL** elements and returns to zero.

TR: Test with **NULL** elements interspersed in the *list*.

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

10 **IF** *PCTS_lio_listio* **THEN**

TEST: A successful call to the *lio_listio*() function, where the *aio_lio_opcode* field of an *aiocb* structure specifies the LIO_NOP operation, causes the *list* entry to be ignored.

TR: Test with multiple file descriptors and buffers in a single call with LIO_NOP operations.

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

lio_read_op

IF *PCTS_lio_listio* **THEN**

TEST: A successful call to the *lio_listio*() function, where the *aio_lio_opcode* field of an *aiocb* structure specifies the LIO_READ operation, causes an I/O operation to be submitted as if by a call to *aio_read* with the *aiocb* structure.

TR: Test with multiple file descriptors and buffers in a single call. If *POSIX_PRIORITY_SCHEDULING*, test with multiple scheduling priorities.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

lio_write_op
IF *PCTS_lio_listio* **THEN**
TEST: A successful call to the *lio_listio*() function where the *aio_lio_opcode* field of an *aiocb* structure specifies the LIO_WRITE operation causes an I/O operation to be submitted as if by a call to *aio_write* with the *aiocb* equal to the address of the *aiocb* structure.
TR: Test with multiple file descriptors and buffers in a single call. If *POSIX_PRIORITY_SCHEDULING*, test with multiple scheduling priorities.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

R_1 **IF** *PCTS_lio_listio* **THEN**
TEST: The *list* element further describes the I/O operation to be performed in a manner identical to that of the corresponding *aiocb* structure when used by the *aio_read*() and *aio_write* () functions.
ELSE *NO_OPTION*
SEE: The lio_read_op and lio_write_op assertions.

11 **IF** *PCTS_lio_listio* and {_POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by calling *open*() with O_RSYNC and O_DSYNC set in the *oflag* parameter.
TEST: A read operation initiated by calling *lio_listio*() either completes by transferring an image of the data to the requesting process, or if unsuccessful, by diagnosing and returning an indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityRead in 2.2.2.119.
Conformance for lio_listio: PASS, NO_TEST, NO_OPTION

12 **IF** *PCTS_lio_listio* and {_POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by calling *open*() with O_RSYNC and O_SYNC set in the *oflag* parameter.
TEST: At the time that the synchronized read operation initiated by calling *lio_listio*() occurs, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data, and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode
2. File serial number
3. ID of device containing this file
4. Number of links
5. User ID of the owner of the file
6. Group ID of the group of the file
7. The file size in bytes
8. Time of last access
9. Time of last data modification

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10. Time of last file status change.

TR: Test for regular files .
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIOFileIntegrityRead in 2.2.2.120.
Conformance for lio_listio: PASS, NO_TEST, NO_OPTION

13 **IF** *PCTS_lio_listio* and {_POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by calling *open*() with O_DSYNC set in the *oflag* parameter.
TEST: A write operation initiated by calling *lio_listio*() either completes by transferring an image of the data to the physical medium containing the file or, if unsuccessful, by diagnosing and returning an indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWrite in 2.2.2.119.
Conformance for lio_listio: PASS, NO_TEST, NO_OPTION

14 **IF** *PCTS_lio_listio* and {_POSIX_SYNCHRONIZED_IO} **THEN** *PCTS_lio_listio* and {_POSIX_SYNCHRONIZED_IO}
SETUP: Open a file by calling *open*() with O_SYNC set in the *oflag* parameter.
TEST: At the time that the synchronized write operation initiated by calling *lio_listio*() occurs, the data are written to the physical medium containing the file, and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode
2. File serial number
3. ID of device containing this file
4. Number of links
5. User ID of the owner of the file
6. Group ID of the group of the file
7. The file size in bytes
8. Time of last access
9. Time of last data modification
10. Time of last file status change.

TR: Test for regular files.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIOFileIntegrityWrite in 2.2.2.120.
Conformance for lio_listio: PASS, NO_TEST, NO_OPTION

6.7.4.3 Returns

R_2 **IF** *PCTS_lio_listio* **THEN**
TEST: A call to the *lio_listio*() function, where the *mode* argument has the value LIO_NOWAIT, returns the value zero and queues the I/O operations.
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SEE: Assertions in 6.7.4.2.

R_3 IF *PCTS_lio_listio* **THEN**
TEST: An unsuccessful call to the *lio_listio*() function where, the *mode* argument has the value LIO_NOWAIT, returns the value -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: Assertions in 6.7.4.4.

R_4 IF *PCTS_lio_listio* **THEN**
TEST: A successful call to the *lio_listio*() function, where the *mode* argument has the value LIO_WAIT, returns the value zero when all the indicated I/O operations have completed successfully.
ELSE *NO_OPTION*
SEE: Assertions in 6.7.4.2.

R_5 IF *PCTS_lio_listio* **THEN**
TEST: An unsuccessful call to the *lio_listio*() function, where the *mode* argument has the value LIO_WAIT, returns the value -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: Assertions in 6.7.4.4.

15 IF *PCTS_lio_listio* **THEN**
TEST: A successful call to the *lio_listio*() function where some individual requests fail does not prevent completion of any other individual request and returns 0.
TR: Intersperse individual I/O operations that will fail with those that will succeed.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

R_6 IF *PCTS_lio_listio* **THEN**
TEST: The error statuses returned by a call to *lio_listio*(), with the *aio_lio_opcode* field of an *aiocb* structure equal to LIO_READ or LIO_WRITE, are identical to those returned as the result of an *aio_read*() or *aio_write*() function, respectively.
ELSE *NO_OPTION*
SEE: Assertions starting with lio_read and lio_write in 6.7.4.4.

6.7.4.4 Errors

16 IF *PCTS_lio_listio* **THEN**
SETUP: Queue enough asynchronous I/O operations before calling *lio_listio*() to perform the test so that not all I/O requests in the call will have a resource to be queued.
TEST: A call to the *lio_listio*() function returns -1 and sets *errno* to [EAGAIN].
TR: Test with both LIO_READ and LIO_WRITE operations.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST, NO_OPTION

17 IF *PCTS_lio_listio* **THEN**
IF {AIO_MAX}≤ *PCTS_AIO_MAX* **THEN**
SETUP: Queue {AIO_MAX} or fewer asynchronous I/O operations before calling *lio_listio*() to perform the test.
TEST: A call to the *lio_listio*() function returns -1 and sets *errno* to [EAGAIN].
TR: Test with both {AIO_MAX} and fewer queued operations. When testing with less than {AIO_MAX} operations queued, make sure that some, but not all, of the I/O operations in the call can be queued. Test with both LIO_READ and LIO_WRITE operations.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_lio_listio* **THEN**
IF {AIO_MAX}≤ *PCTS_AIO_MAX* and *PCTS_GTI_DEVICE* **THEN**
SETUP: Queue {AIO_MAX} asynchronous I/O operations before calling *lio_listio*() for the test.
TEST: A call to the *lio_listio*() function returns -1 and sets *errno* to [EAGAIN].
TR: Test for a character special device. Test with both {AIO_MAX} and fewer queued operations. When testing with less than {AIO_MAX} operations queued, make sure that some, but not all, of the I/O operations in the call can be queued. Test with both LIO_READ and LIO_WRITE operations.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_lio_listio* **THEN**
IF {AIO_MAX}> *PCTS_AIO_MAX* and *PCTS_GTI_DEVICE* **THEN**
SETUP: Queue *PCTS_AIO_MAX* - *nent* asynchronous I/O operations before calling *lio_listio*().
TEST: A call to the *lio_listio*() function with *nent* entries returns 0 and does not set *errno* to [EAGAIN].
TR: Test for a character special device. Test with both LIO_READ and LIO_WRITE operations.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

lio_read_ebadf1
IF *PCTS_lio_listio* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *lio_listio*() function, where *list[]->aio_lio_opcode* is equal to LIO_READ and *list[]->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously, then the *lio_listio*() function returns -1 and sets *errno* to [EIO]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EIO] .
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

lio_read_ebadf2
IF *PCTS_lio_listio* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *lio_listio*() function, where *list[]->aio_lio_opcode* is equal to LIO_READ and *list[]->aio_fildes* argument is not open for reading, the implementation either detects the condition synchronously, then the *lio_listio*() function returns -1 and sets *errno* to [EIO]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EIO] .
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

lio_read_einval1
IF *PCTS_lio_listio* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: After a call to the *lio_listio*() function, where *list[]->aio_lio_opcode* is equal to LIO_READ and the file offset value implied by *list[]->aio_offset* would be invalid, or *list[]->aio_reqprio* is not a valid value, or *list[]->aio_nbytes* is an invalid value, the implementation either detects the condition synchronously, then the *lio_listio*() function returns -1 and sets *errno* to [EIO]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the the asynchronous operation is set to [EIO].

TR: Test separately for each of the three conditions above.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

20 **IF** *PCTS_lio_listio* and {_POSIX_JOB_CONTROL} **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Make the testing process part of the background process group and make the process group of the process orphaned. Call to the *lio_listio*() function with *list[]->aio_lio_opcode* equal to LIO_READ, so that it successfully queues the I/O operation to read from its controlling terminal.

TEST: The read operation will fail; the return status of the asynchronous operation returned by a call to the *aio_return* () function is -1 and the error status returned by a call to the *aio_error*() function is [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **IF** *PCTS_lio_listio* and {_POSIX_JOB_CONTROL} **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Make the testing process part of the background process group and have the process ignore or block the SIGTTIN signal. Call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_READ, so that it successfully queues the I/O operation to read from its controlling terminal.

TEST: The read operation will fail; the return status of the asynchronous operation returned by a call to the *aio_return* () function is -1 and the error status returned by a call to the *aio_error*() function is [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

R_7 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function, where *list[]->aio_lio_opcode* is equal to LIO_READ and the *list[]->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously: the *lio_listio*() function returns -1 and sets *errno* to [EIO]; or it detects the condition asynchronously: the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_read_ebadf1 in 6.7.4.4.

R_8 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* **THEN**

TEST: After a call to the *lio_listio*() function, where *list[]->aio_lio_opcode* is equal to LIO_READ and the *list[]->aio_fildes* argument is not open for reading, the implementation either detects the condition synchronously, then the

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

lio_listio() function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_read_ebadf2 in 6.7.4.4.

22 **IF** *PCTS_lio_listio* and *PCTS_aio_cancel* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Call to the *lio_listio*() function with a *list[]->aio_lio_opcode* equal to LIO_READ and so that it successfully queues the I/O operation.

TEST: A read operation, that is canceled after a call to the *aio_cancel*() function and before the I/O completed, results in a return status from a call to the *aio_return*() function of -1 and the error status from a call to the *aio_error*() function of [ECANCELED].

TR: Test for both a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

R_9 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: After a call to the *lio_listio*() function, where *list[]->aio_lio_opcode is* equal to LIO_READ and the file offset value implied by *list[]->aio_offset* would be invalid, the implementation detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EINVAL].

TEST: The read operation will fail and the return status of the asynchronous operation returned by a call to the *aio_return* () function is -1 and the error status returned by a call to the *aio_error*() function is [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_read_einval in 6.7.4.4.

lio_write_ebadf1

IF *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function, where *list[]->aio_lio_opcode* is equal to LIO_WRITE and the *list[]->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously, then the *lio_listio*() function returns -1 and sets *errno* to [EIO]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EIO] .

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

lio_write_ebadf2

IF *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function where *list[]->aio_lio_opcode* is equal to LIO_WRITE and the *list[]->aio_fildes* argument is not open for writing, the implementation either detects the condition synchronously, then the *lio_listio*() function returns -1 and sets *errno* to [EIO]; or it detects the condition asynchronously, then the return status of the asynchronous

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

operation is set to -1 and the error status of the asynchronous operation is set to [EIO] .

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

lio_write_einval1

IF *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function, where *list[]->aio_lio_opcode* is equal to LIO_WRITE and file offset value implied by *list[]->aio_offset* would be invalid, or the *list[]->aio_reqprio* is not a valid value, or *list[]->aoi_nbytes* is an invalid value, the implementation either detects the condition synchronously, then the *lio_listio*() function returns -1 and sets *errno* to [EIO]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to [EIO] .

TR: Test separately for each of the three conditions above.

If *PCTS_GTI_DEVICE*, test for a character special file.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

23 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_cancel* and *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_WRITE and so that it successfully queues the I/O operation.

TEST: After a write operation is canceled by a call to the *aio_cancel*() function and no data were transferred, the return status of the asynchronous operation returned by a call to the *aio_return*() function is -1 and the error status returned by a call to the *aio_error*() function is [EINTR] or [ECANCELED].

TR: Test for both a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

24 **IF** *PCTS_lio_listio* **THEN**

IF the implementation supports a maximum file size **THEN**

TEST: An attempt to write a file that would exceed an implementation-defined maximum size by calling the *lio_listio*() function, where the write operation is successfully queued, causes the error status for the asynchronous operation to be [EFBIG].

NOTE: The assertion test would require an unreasonable amount of time or resources for most implementations.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

25 **IF** *PCTS_lio_listio* and {_POSIX_JOB_CONTROL} **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Make the testing process part of the background process group; set TOSTOP; ignore or block SIGTTOU signals; and make the process group of the process orphaned. Call the *lio_listio*() function so that it successfully queues the I/O operation to write from its controlling terminal.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: The write operation will fail; the return status of the asynchronous operation returned by a call to the *aio_return*() function is -1 and the error status returned by a call to the *aio_error*() function is [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

26 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* and *PCTS_DETECT_ENOSPC* **THEN**

SETUP: Fill a device so that there is no more space available on it for data. Call the *lio_listio*() function so that it successfully queues a write I/O operation.

TEST: After a call to the *lio_listio*() function the implementation either detects the condition synchronously, the *lio_listio*() function returns -1 and sets *errno* to [EIO]; or it detects the condition asynchronously, the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

27 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Call the *lio_listio*() function so that it successfully queues a write I/O operation.

TEST: After a call to the *lio_listio*() function to write to a pipe or FIFO that is not open for reading by any process, the implementation either detects the condition synchronously, the *lio_listio*() function returns -1 and sets *errno* to [EIO]; or it detects the condition asynchronously, the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

R_10 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function, where the *list[]->aio_fildes* argument is not a valid file descriptor, the implementation detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_write_ebadf1 in 6.7.4.4.

R_11 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* **THEN**

TEST: After a call to the *lio_listio*() function, where the *list[]->aio_fildes* argument is not open for writing, the implementation either detects the condition synchronously, then the *lio_listio*() function returns -1 and sets *errno* to [EBADF]; or it detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_write_ebadf2 in 6.7.4.4.

28 **IF** *PCTS_lio_listio* and *PCTS_aio_cancel* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Call the *lio_listio*() function so that it successfully queues a write I/O operation.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: A write function, that is canceled after a call to the *aio_cancel*() function and before the I/O operation is completed, results in a return status returned by a call to the *aio_return*() function that is -1 and the error status returned by a call to the *aio_error*() function that is [ECANCELED].

TR: Test for both a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

R_12 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function for a write operation, where the file offset value implied by *list[]->aio_offset* would be invalid, the implementation detects the condition asynchronously, then the return status of the asynchronous operation is set to -1 and the error status of the asynchronous operation is set to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_write_einval1 in 6.7.4.4.

29 **IF** *PCTS_lio_listio* **THEN**

TEST: A call to the *lio_listio*() where the *mode* argument is not a proper value returns -1 and sets *errno* to [EINVAL].

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

30 **IF** *PCTS_lio_listio* **THEN**

TEST: A call to the *lio_listio*() where the value of *nent* is greater than {AIO_LISTIO_MAX}, returns -1 and sets *errno* to [EINVAL].

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

31 **IF** *PCTS_lio_listio* **THEN**

SETUP: Call to *lio_listio*(), where the *mode* argument is LIO_WAIT and where there are at least two I/O operations, at least one of which will complete before the others and send a signal indicating its completion.

TEST: Such *lio_listio*() call will receive a signal while waiting for all I/O requests to complete during an operation, return -1, set *errno* to [EINTR], and outstanding I/O requests will not be canceled.

TR: Test for a signal also generated by another process.

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

32 **IF** not *PCTS_lio_listio* **THEN**

TEST: A call to the *lio_listio*() function returns -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

6.7.5 Retrieve Error Status of Asynchronous I/O Operation

Function: *aio_error*().

6.7.5.1 Synopsis

1

M_GA_stdC_proto_decl(int; aio_error; const struct aiocb * aiocbp; aio.h;;;)

SEE: Assertion GA_stdC_proto_decl in 2.7.3.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for aio_error: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(aio_error; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for aio_error: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; aio_error; aio.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for aio_error: PASS, NO_OPTION

4

M_GA_macro_args (aio_error; aio.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for aio_error: PASS, NO_OPTION

6.7.5.2 Description

5 **IF** *PCTS_aio_error* **THEN**
TEST: A call to the *aio_error*() function returns the error status associated with the *aiocb* structure referenced by the *aiocbp* argument.
ELSE *NO_OPTION*
Conformance for aio_error: PASS, NO_OPTION

R_1 **IF** *PCTS_aio_error* **THEN**
TEST: The error status returned by a call to *aio_error*() is the *errno* value that would be set by the corresponding *read*(), *write*(), or *fsync*() operation.
ELSE *NO_OPTION*
SEE: Assertions for *aio_read*() in 6.7.2.4.

6 **IF** *PCTS_aio_error* **THEN**
TEST: A call to the *aio_error*() function where the *aiocb* structure referenced by the *aiocbp* argument refers to an operation that has not yet completed returns the value [EINPROGRESS].
ELSE *NO_OPTION*
Conformance for aio_error: PASS, NO_OPTION

6.7.5.3 Returns

7 **IF** *PCTS_aio_error* **THEN**
TEST: A call to the *aio_error*() function, where the *aiocb* structure referenced by the *aiocbp* argument refers to an asynchronous I/O operation that has completed successfully, returns 0.
ELSE *NO_OPTION*
Conformance for aio_error: PASS, NO_OPTION

R-2 **TEST:** The error status returned by a call to *aio_error*(), for an asynchronous I/O operation that has completed unsuccessfully, is the *errno* value that would be set by the corresponding *read*(), *write*(), or *fsync*() operation.
ELSE *NO_OPTION*
SEE: Assertions for *aio_read*() in 6.7.2.4.

8 **IF** *PCTS_aio_error* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: A call to the *aio_error*() function, where the *aiocb* structure referenced by the *aiocbp* argument refers to an asynchronous I/O operation that has not yet completed, returns [EINPROGRESS].

ELSE *NO_OPTION*

Conformance for aio_error: PASS, NO_OPTION

6.7.5.4 Errors

9 **IF** not *PCTS_aio_error* **THEN**

TEST: A call to the *aio_error*() returns -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for aio_error: PASS, NO_OPTION

10 **IF** *PCTS_aio_error* **THEN**

IF *PCTS_DETECT_AIO_ERROR_AIOCBP* **THEN**

TEST: A call to the *aio_error*(), where the *aiocbp* argument refers to an asynchronous operation whose return status has previously been retrieved, returns -1 and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_error: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_aio_error* **THEN**

IF *PCTS_DETECT_AIO_ERROR_AIOCBP* **THEN**

TEST: A call to the *aio_error*() function, where the *aiocbp* argument refers to an invalid *aiocb* structure, returns -1 and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_error: PASS, NO_OPTION

6.7.6 Retrieve Return Status of Asynchronous I/O Operation

Function: *aio_return*().

6.7.6.1 Synopsis

1

M_GA_stdC_proto_decl(int; aio_return; struct aiocb * aiocbp; aio.h;;;)

SEE: Assertion GA_stdC_proto_decl in 2.7.3.

Conformance for aio_return: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(ssize_t; aio_return; aio.h;;;)

SEE: Assertion GA_commonC_result_decl in 2.7.3.

Conformance for aio_return: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(ssize_t; aio_return; aio.h;;;)

SEE: Assertion GA_macro_result_decl in 1.3.4.

Conformance for aio_return: PASS, NO_OPTION

4

M_GA_macro_args (aio_return; aio.h;;;)

SEE: Assertion GA_macro_args in 2.7.3.

Conformance for aio_return: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.7.6.2 Description
return_status

IF ***PCTS_aio_return*** **THEN**
TEST: A call to the *aio_return*() function returns the return status associated with the *aiocb* structure referenced by the *aiocbp* argument.
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

R_1 IF *PCTS_aio_return* **THEN**
TEST: The return status for an asynchronous I/O operation is the value that would be returned by the corresponding *read*(), *write*(), or *fsync*() function call.
ELSE *NO_OPTION*
SEE: Assertions for *aio_read*() in 6.7.2.4.

D_1 IF *PCTS_aio_return* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the return status for an operation returned by a call to *aio_return*(), when the error status is equal to [EINPROGRESS], does so in 6.7.6.2.
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

R_2 IF *PCTS_aio_return* **THEN**
SETUP: Initiate an asynchronous I/O operation and let it complete. Retrieve the return status by calling *aio-return*().
TEST: A call to the *aio_return*() or the *aio_error*() function using the same *aiocb* structure as used in a previously successful call to *aio_return*() returns -1 and sets *errno* to [EINVAL].
TR: Test for calling both *aio_return*() and *aio_error*().
ELSE *NO_OPTION*
SEE: Assertion einval1 in 6.7.6.4.

5 IF *PCTS_aio_return* **THEN**
SETUP: Initiate and complete an asynchronous I/O operation. Call the *aio_return*() function to retrieve its return status. Submit the same *aiocb* structure referred to by *aiocbp* in the *aio_return*() call to another asynchronous operation.
TEST: A call to the *aio_return*() function can reuse an *aiocb* structure used in a previously successful *aio_return*() call after it has been used to initiate another asynchronous I/O operation.
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

6.7.6.3 Returns

R_3 IF *PCTS_aio_return* **THEN**
SETUP: After an asynchronous I/O operation has completed, the return status, as described for *read*(), *write*(), and *fsync*() , is returned.
ELSE *NO_OPTION*
SEE: Assertion return_status in 6.7.6.4.

D_2 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the results of a call to the *aio_return*() function, when the asynchronous I/O operation has not yet completed, does so in 6.7.6.4.
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

6.7.6.4 Errors

einval1

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

IF *PCTS_aio_return* **THEN**
SETUP: Initiate an asynchronous I/O operation and let it complete. Retrieve the return status by calling *aio_return*().
TEST: A call to the *aio_return*() or the *aio_error*() function, using the same *aiocb* structure as used in a previously successful call to *aio_return*(), returns -1 and sets *errno* to [EINVAL].
TR: Test for calling *aio_return*(). If *PCTS_DETECT_AIO_ERROR_AIOCBP* then test for *aio_error*().
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

6 **IF** *PCTS_aio_return* **THEN**
TEST: A call to the *aio_return*() function, where the *aiocbp* argument refers to an invalid *aiocb* structure, returns -1 and sets *errno* to [EINVAL].
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

7 **IF** not *PCTS_aio_return* **THEN**
TEST: A call to the *aio_return*() function returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

6.7.7 Cancel Asynchronous I/O Request

Function: *aio_cancel*().

6.7.7.1 Synopsis

1
M_GA_stdC_proto_decl(int; aio_cancel; int fildes, struct aiocb * aiocbp; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for aio_cancel: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(aio_cancel; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for aio_cancel: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; aio_cancel; aio.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for aio_cancel: PASS, NO_OPTION

4
M_GA_macro_args (aio_cancel; aio.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for aio_cancel: PASS, NO_OPTION

6.7.7.2 Description

5 **IF** *PCTS_aio_cancel* **THEN**
IF *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**
SETUP: Queue asynchronous I/O requests for one file descriptor that will not complete before calling the *aio_cancel*() function.
TEST: A call to the *aio_cancel*() function will cancel the asynchronous I/O request currently outstanding against file descriptor *fildes* specified by the *aiocbp* argument and return the value AIO_CANCELED.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TR: Test by queuing a single request. Then test by queueing multiple requests, each of which uses a different *aiocbp* against a single file descriptor. Then test by queueing multiple requests against multiple file descriptors.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_cancel: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** *PCTS_aio_cancel* **THEN**

IF *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**

SETUP: Queue multiple asynchronous I/O requests for multiple file descriptors that will not complete signal notification before calling the *aio_cancel*() function.

TEST: A call to the *aio_cancel*() function, where the *aiocbp* argument is **NULL**, cancels the outstanding cancellable asynchronous I/O requests against the file descriptor *fildes*, returns the value AIO_CANCELED, and normal signal delivery occurs for the canceled operations.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_cancel: PASS, NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_aio_cancel* **THEN**

IF *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**

TEST: Any asynchronous I/O requests that cannot be canceled by a call to the *aio_cancel*() function complete following the normal asynchronous completion process.

NOTE: There is no known portable test method for this assertion.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_cancel: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_aio_cancel* **THEN**

IF *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**

TEST: After a call to the *aio_cancel*() function, for requested operations that are successfully canceled, the associated error status is set to [ECANCELED] and the return status is -1.

TR: Test for the *aiocbp* pointing to a valid *aiocb* and for it being **NULL**.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_cancel: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_aio_cancel* **THEN**

IF *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**

TEST: After a call to the *aio_cancel*() function, for requested operations that are not successfully canceled, the *aiocbp* is not modified by *aio_cancel*().

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_cancel: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_aio_cancel* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the results of calling *aio_cancel*(), where the *aiocbp* is not **NULL** and the *fildes* argument does not have the same value as the file descriptor with which the asynchronous operation was initiated, does so in 6.7.7.2.

ELSE *NO_OPTION*

Conformance for aio_cancel: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D_2 IF *PCTS_aio_cancel* **THEN**
TEST: The PCD.1b that documents which asynchronous I/O operations are cancellable by calling the *aio_cancel*() function, does so in 6.7.7.2.
ELSE *NO_OPTION*
Conformance for aio_cancel: PASS, NO_OPTION

6.7.7.3 Returns

R_1 IF *PCTS_aio_cancel* **THEN**
TEST: The *aio_cancel*() function returns the value AIO_CANCELED to the calling process if the requested operation(s) were canceled.
ELSE *NO_OPTION*
SEE: Assertions in 6.7.7.2.

10 IF *PCTS_aio_cancel* **THEN**
IF *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**
TEST: After a call to the *aio_cancel*() function, the value AIO_NOTCANCELED is returned when at least one of the requested operation(s) cannot be canceled because it is in progress.
NOTE: There is no known portable test method for assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_cancel: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

11 IF *PCTS_aio_cancel* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
TEST: The value AIO_ALLDONE is returned for a call to the *aio_cancel*() function when all of the operations have already completed before they could be canceled.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_cancel: PASS, NO_TEST_SUPPORT, NO_OPTION

R_2 IF *PCTS_aio_cancel* **THEN**
TEST: When an error occurs in a call to the *aio_cancel*() function, it returns -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: Assertions in 6.7.7.4.

6.7.7.4 Errors

12 IF *PCTS_aio_cancel* **THEN**
IF *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**
TEST: A call to the *aio_cancel*() function, where the *fildes* argument is not a valid file descriptor, returns -1 and sets *errno* to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_cancel: PASS, NO_OPTION

13 IF not *PCTS_aio_cancel* **THEN**
TEST: A call to the *aio_cancel*() function returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for aio_cancel: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.7.8 Wait for Asynchronous I/O Request

6.7.9.2 De: *11.2.3 Initia* Func

6.7.8.1 Synopsis

1

M_GA_stdC_proto_decl(int; aio_suspend; const struct aiocb * const list[], int nent, const struct timespec *timeout; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for aio_suspend: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(aio_suspend; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for aio_suspend: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; aio_suspend; aio.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for aio_suspend: PASS, NO_OPTION

4

M_GA_macro_args (aio_suspend; aio.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for aio_suspend: PASS, NO_OPTION

6.7.8.2 Description

completion

IF *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous I/O operations that will not send signals when they complete and that will not complete until after *aio_suspend*() has been called.
TEST: A call to the *aio_suspend*() function, with the *timeout* argument equal to **NULL**, suspends the calling process until at least one of the asynchronous I/O operations referenced by the *list* argument has completed and returns 0.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_suspend: PASS, NO_TEST_SUPPORT, NO_OPTION

interrupt

IF *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous I/O operations that will send signals when they complete and that will not complete until after *aio_suspend*() has been called.
TEST: A call to the *aio_suspend*() function, with the *timeout* argument equal to **NULL**, suspends the calling process until a signal interrupts the function and returns -1 and sets *errno* to [EINTR].
TR: Test for a signal coming from the completion of an asynchronous I/O operation and for a signal coming from another process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_suspend: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

timeout

IF *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous I/O operations that will not complete; pass the list of the operations to the *aio_suspend*() function in the test.
TEST: A call to the *aio_suspend*() function, with the *timeout* argument not equal to **NULL**, suspends the calling process until the time interval specified by *timeout* has elapsed, returns -1, and sets *errno* to [EAGAIN].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_suspend: PASS, NO_TEST_SUPPORT, NO_OPTION

already_completed

IF *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous I/O operations so that at least one of them will complete before *aio_suspend*() is called.
TEST: A call to the *aio_suspend*() function, where any of the *aiocb* structures in the *list* argument correspond to completed asynchronous I/O operations (i.e., the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, returns without suspending the calling process and returns 0.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_suspend: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
TEST: A call to the *aio_suspend*() function ignores any **NULL** pointers in the *list* argument.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_suspend: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** a PCD.1b documents the following THEN
TEST: A PCD.1b that documents the effect of a call to the *aio_suspend*() function, where the *list* argument contains pointers that refer to *aiocb* structures that have not been used in submitting asynchronous I/O, does so in 6.7.8.2.
ELSE *NO_OPTION*
Conformance for aio_suspend: PASS, NO_OPTION

R_1 **IF** *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
TEST: After a call to the *aio_suspend*() function, and after the time interval indicated in the *timespec* structure, pointed to by *timeout,* elapses before any of the I/O operations referenced by *list* are completed, the *aio_suspend*() returns with an error.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion timeout in 6.7.8.2.

6.7.8.3 Returns

R_2 **IF** *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
TEST: A call to the *aio_suspend*() function, after one or more asynchronous I/O operations have completed, returns 0.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertions completion and already_completed in 6.7.8.2.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

R_3 **IF** *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
TEST: A call to the *aio_suspend*() function returns a value of -1 and sets *errno* to indicate the error, when an error condition is detected.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertions timeout and interrupt in 6.7.8.2 and no_support in 6.7.8.4.

6.7.8.4 Errors

R_4 **IF** *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
TEST: A call to the *aio_suspend*(), where no asynchronous I/O indicated in the list referenced by *list* completed in the time interval indicated by *timeout*, returns -1 and sets *errno* to [EAGAIN].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion timeout in 6.7.8.2.

6 **TEST:** A call to *aio_suspend*(), where a signal interrupts the *aio_suspend*() function, returns -1 and sets *errno* to [EINTR].
SEE: Assertion interrupt in 6.7.8.2.
Conformance for aio_suspend: PASS

no_support
IF not *PCTS_aio_suspend* THEN
TEST: A call to the *aio_suspend*() function returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for aio_suspend: PASS, NO_OPTION

6.7.9 Asynchronous File Synchronization

Function: *aio_fsync*()

6.7.9.1 Synopsis

1
M_GA_stdC_proto_decl(int; aio_fsync; int op, struct aiocb * aiocbp; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for aio_fsync: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(aio_fsync; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for aio_fsync: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; aio_fsync; aio.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for aio_fsync: PASS, NO_OPTION

4
M_GA_macro_args (aio_fsync; aio.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for aio_fsync: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.7.9.2 Description

5 **IF** *PCTS_aio_fsync* **THEN**
IF *PCTS_aio_read* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous read operations using as many of *PCTS_aio_read* and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O data synchronization has not been set.
TEST: A call to the *aio_fsync*() function, with the *op* argument equal to O_DSYNC, asynchronously forces all read operations associated with the file indicated by the file descriptor *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*() to the synchronized I/O data completion state and returns 0 when the synchronization request has been initiated or queued to the file or device. That is, the read operation either completes by transferring an image of the data to the requesting process or, if unsuccessful, by diagnosing and returning an indicator of the error.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityRead in 2.2.2.119.
Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** *PCTS_aio_fsync* **THEN**
IF *PCTS_aio_read* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous read operations using as many of *PCTS_aio_read* and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O data synchronization has not been set.
TEST: A call to the *aio_fsync*() function with the *op* argument equal to O_DSYNC asynchronously forces all read operations associated with the file indicated by the file descriptor *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*() to the synchronized I/O data completion state and returns 0 when the synchronization request has been initiated or queued to the file or device. That is, at the time that the synchronized read operation initiated by calling *aio_fsync*() any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWbeforeR in 2.2.2.119.
Conformance for aio_fsync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_aio_fsync* **THEN**
IF *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous write operations using as many of *PCTS_aio_write* and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O data synchronization has not been set.
TEST: A call to the *aio_fsync*() function with the *op* argument equal to O_DSYNC asynchronously forces all write operations associated with the file indicated by the file descriptor *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*() to the synchronized I/O data completion state and returns 0 when the synchronization request has been initiated or queued to the file or device. That is, a write operation initiated by calling *aio_fsync*() either completes by transferring an image of the data to the physical medium containing the file or, if unsuccessful, by diagnosing and returning and indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWrite in 2.2.2.119.
Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8 **IF** *PCTS_aio_fsync* **THEN**
IF *PCTS_aio_read* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous read operations using as many of *PCTS_aio_read* and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O data synchronization has not been set.
TEST: A call to the *aio_fsync*() function with the *op* argument equal to O_SYNC asynchronously forces all read operations associated with the file indicated by the file descriptor *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*() to the synchronized I/O data completion state and returns 0 when the synchronization request has been initiated or queued to the file or device. That is: At the time that the synchronized read operation initiated by calling *aio_fsync*() occurs, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode
2. File serial number
3. ID of device containing this file
4. Number of links
5. User ID of the owner of the file
6. Group ID of the group of the file
7. The file size in bytes
8. Time of last access
9. Time of last data modification
10. Time of last file status change.

NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIOFileIntegrityRead in 2.2.2.120.
Conformance for aio_fsync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_aio_fsync* **THEN**
IF *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous write operations using as many of *PCTS_aio_write* and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O data synchronization has not been set.
TEST: A call to the *aio_fsync*() function with the *op* argument equal to O_SYNC asynchronously forces all write operations associated with the file indicated by the file descriptor *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*() to the synchronized I/O file completion state and returns 0 when the synchronization request has been initiated or queued to the file or device. That is: At the time that the synchronized write operation initiated by calling *aio_fsync*() occurs, the data are written to the physical medium containing the file and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2. File serial number
3. ID of device containing this file
4. Number of links
5. User ID of the owner of the file
6. Group ID of the group of the file
7. The file size in bytes
8. Time of last access
9. Time of last data modification
10. Time of last file status change.

NOTE: There is no known portable test method for this assertion.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion GA_syncIOFileIntegrityWrite in 2.2.2.120.

Conformance for aio_fsync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

10 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_aio_read* or *PCTS_write* or *PCTS_lio_listio* **THEN**

TEST: After a call to the *aio_fsync*() function and after the request is queued, the error status for all asynchronous I/O operations associated with the file indicated by the file descriptor *aio-fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call will be [EINPROGRESS], and the *aio-fsynch*() call returns 0.

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write,* and *PCTS_lio_listio* as are supported by the implementation.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_aio_read* or *PCTS_write* or *PCTS_lio_listio* **THEN**

TEST: After a call to the *aio_fsync*() function and after all data has been successfully transferred, the error status of each queued I/O operation is reset to reflect the success or failure of the operation.

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write,* and *PCTS_lio_listio* as are supported by the implementation. Test for operations that succeed and that fail.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

12 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_aio_read* or *PCTS_write* or *PCTS_lio_listio* **THEN**

SETUP: Queue asynchronous operations, including some write operations, using as many of *PCTS_aio_read, PCTS_aio_write,* and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O synchronization has not been set.

TEST: After a call to the *aio_fsync*() function where the *aio-sigevent.sigev_signo* is nonzero, a signal is generated when all operations have achieved synchronized I/O completion, and the *aio_fsync*() call returns 0.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNCH.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_aio_read* or *PCTS_write* or *PCTS_lio_listio* **THEN**

SETUP: Queue asynchronous operations, including some write operations using as many of *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O synchronized has not been set.

TEST: During a call to the *aio_fsync*() function all members of the structure referenced by *aiocbp* are ignored except for *aio_fildes* and *aio_sigevent.*

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNC.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_aio_fsync* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the behavior of an implementation when the control block referenced by *aiocbp* in a call to the *aio_fsync*() function becomes an illegal address prior to asynchronous I/O completion does so in 6.7.9.2.

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_OPTION

NOTE: The following statement in POSIX.1b {3} does not make sense since there is no file to synchronize:

If *aiocbp* is NULL, then no status is returned in *aiocbp*, and no signal is generated upon completion of the operation.

6.7.9.3 Returns

R_1 **IF** *PCTS_aio_fsync* **THEN**

TEST: The *aio_fsync*() function returns the value 0 to the calling process when the I/O operation is successfully queued.

ELSE *NO_OPTION*

SEE: Assertions in 6.7.9.2.

R_2 **IF** *PCTS_aio_fsync* **THEN**

TEST: The *aio_fsync*() function returns the value -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: Assertions in 6.7.9.4.

6.7.9.4 Errors

14 **IF** *PCTS_aio_fsync* **THEN**

IF *{AIO_MAX}* $\leq$ *PCTS_AIO_MAX* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio)* **THEN**

SETUP: Queue {AIO_MAX} asynchronous I/O operations that will not complete for a file that does not have its synchronized I/O attributes set.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: A call to the *aio_fsync*() function, when the requested asynchronous operation was not queued due to temporary resource limitations, returns -1 and sets *errno* to [EAGAIN].

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write,* and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNC.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_aio_fsync* **THEN**

IF *{AIO_MAX}* ≤ *PCTS_AIO_MAX* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio)* **THEN**

SETUP: Queue *PCTS_AIO_MAX* asynchronous I/O operations that will not complete for a file that does not have its synchronized I/O attributes set.

TEST: A call to the *aio_fsync*() returns 0 and does not set *errno* to [EAGAIN].

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write,* and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNC.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**

TEST: A call to the *aio_fsync*(), where the *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument is not a valid file descriptor open for writing, returns -1 and sets *errno* to [EBADF].

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write,* and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNC.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

17 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_NO_SYNCH_IO_FILE* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio)* **THEN**

SETUP: Queue asynchronous I/O operations for a file for which the implementation does not support synchronized I/O.

TEST: A call to the *aio_fsync*() function, for a file for which the implementation does not support synchronized I/O, returns -1 and sets *errno* to [EINVAL].

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write,* and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNC.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_aio_fsync* **THEN**

TEST: A call to the *aio_fsync*() function, with a value of *op* other than O_DSYNC or O_SYNC, returns -1 and sets *errno* to [EINVAL].

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_OPTION

19 **IF** not *PCTS_aio_fsync* **THEN**

TEST: A call to the *aio_fsync*() function returns -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 7: Device- and Class- Specific Functions

180 There are there are no POSIX.1b {3} assertions in Section 7.

--`,,`,,`-`-`,,`,,`,`,,`---
Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 8: Language-Specific Services for the C Programming Language

There are there are no POSIX.1b {3} assertions in Section 8 except for clause 8.2.2.2.

8.2.2 Open a Stream on a File Descriptor

Function: *fdopen*().

8.2.2.1 Synopsis

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

8.2.2.2 Description

D_1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of the *fdopen*() function, when *fildes* refers to a shared memory object, does so in 8.2.2.3.
ELSE *NO_OPTION*
Conformance for fdopen: PASS, NO_OPTION

8.2.2.3 Returns

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

8.2.2.4 Errors

There are only IEEE Std 2003.1-1992 {4} assertions in this clause; there are no POSIX.1b {3} assertions.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 9: System Databases

180 There are there are no POSIX.1b {3} assertions in Section 9.

--`,,`,,`-`-`,,`,,`,`,,`---
Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 10: Data Interchange Format

180 There are there are no POSIX.1b {3} assertions in Section 10.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 11: Synchronization

11.1 Semaphore Characteristics

1 **SETUP:** Include `<semaphore.h>`.
TEST: The type *sem_t* is defined.
Conformance for sem.hdr: PASS

M_GA_semOpenMaxFD(PCTS_SEM_FILE_DESCRIPTORS; function; PCTS_function) =
IF ***PCTS_SEM_FILE_DESCRIPTORS*** **THEN**
IF ***({OPEN_MAX}***<= *PCTS_OPEN_MAX*) and *PCTS_function* **THEN**
TEST: A process calling *function*() can simultaneously open a combination of files and semaphores totaling at least {OPEN_MAX}.
TR: Test for opening {OPEN_MAX} semaphores.
Test for opening a semaphore after opening {OPEN_MAX} -1 files.
Test for opening a file after opening {OPEN_MAX} -1 semaphores.
ELSE *NO_TEST_SUPPORT*
ELSE NO_OPTION

GA_semOpenMaxFD
FOR: *sem_init*() and *sem_open*()
M_GA_semOpenMaxFD(PCTS_SEM_FILE_DESCRIPTORS; function; PCTS_function) =
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
Conformance for sem_hdr: PASS, NO_TEST_SUPPORT, NO_OPTION

M_GA_semPCTSOpenMaxFD(PCTS_SEM_FILE_DESCRIPTORS; function; PCTS_function) =
IF *PCTS_SEM_FILE_DESCRIPTORS* **THEN**
IF ({OPEN_MAX}<= *PCTS_OPEN_MAX*) and *PCTS_function* **THEN**
TEST: A process calling *function*() can simultaneously open a combination of files and semaphores totaling at least *PCTS_OPEN_MAX.*
TR: Test for opening *PCTS_OPEN_MAX*-1 files.
Test for opening a file after opening {OPEN_MAX} -1 files.
Test for opening a file after opening *PCTS_OPEN_MAX* -1 semaphores.
ELSE *NO_TEST_SUPPORT*
ELSE NO_OPTION

GAsemPCTSOpenMaxFD
FOR: *sem_init*() and *sem_open*()
M_GA_semPCTSOpenMaxFD(PCTS_SEM_FILE_DESCRIPTORS; function; PCTS_function)
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
Conformance for sem_hdr: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.2 Semaphore Functions

11.2.1 Initialize an Unnamed Semaphore

Function: *sem_init*().

11.2.1.1 Synopsis

1

M_GA_stdC_proto_decl(int; sem_init; sem_t *sem, int pshared, unsigned int value; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sem_init: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sem_init; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sem_init: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sem_init; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sem_init: PASS, NO_OPTION

4

M_GA_macro_args (sem_init; semaphore.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sem_init: PASS, NO_OPTION

11.2.1.2 Description

M_GA_semOpenMaxFD(PCTS_SEM_INIT_FD; sem_init; PCTS_sem_init)
SEE: Assertion GA_semOpenMaxFD in 11.1.
Conformance for sem_init: PASS/OpenMaxSems, PCTSOpenMaxSems]

M_GA_semPCTSOpenMaxFD(PCTS_SEM_INIT_FD; sem_init; PCTS_sem_init)
SEE: Assertion GA_semPCTSOpenMaxFD in 11.1.
Conformance for sem_init: PASS/OpenMaxSems, PCTSOpenMaxSems]

sem_init

IF *PCTS_sem_init* **THEN**
IF: *PCTS_GAP_sem_init* **THEN**
TEST: A call *sim_init* (*sem, pshared, val)* initializes the unnamed semaphore *sem* to the value of *val.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **FOR:** ***sem_wait*(), *sem_trywait*(), *sem_post*(), *and sem_destroy* ()**
IF *PCTS_sem_init* **THEN**
IF: *PCTS_GAP_sem_init* and *PCTS_function* **THEN**
TEST: The interface *function*() returns successfully when using a semaphore created by *sem_init*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** *PCTS_sem_init* **THEN**
IF: *PCTS_GAP_sem_init* **THEN**
TEST: A semaphore created by *sem_init*() can be used until it is destroyed.
TR: Create a semaphore with a positive value, decrement to zero, then increment and re-decrement to zero.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

7 **FOR:** *sem_wait*(), *sem_trywait*(), *sem_post*(), and *sem_destroy*()
IF *PCTS_sem_init* **THEN**
IF: *PCTS_GAP_sem_init* and *PCTS_function* **THEN**
SETUP: Create a semaphore by calling *sem_init*(*sem, pshared, val*) with a non-zero value of *pshared.*
TEST: Any process that can access the semaphore *sem* can use it for performing *function*().
TR: Test with values {INT_MAX} and {INT_MIN} for *pshared.*
Perform *function*() on the semaphore from the same process that created it and from a child process for each of these values of *pshared.*
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_sem_init* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of referring to copies of *sem* in calls to *sem_wait*(), *sem_trywait*(), *sem_post*(), and *sem_destroy*(), does so in 11.2.1.2.
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_OPTION

D_2 **IF** *PCTS_sem_init* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of a *pshared* argument of zero does so in 11.2.1.2.
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_OPTION

D_3 **IF** *PCTS_sem_init* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sem_init*() function does so in 11.2.1.2.
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_OPTION

11.2.1.3 Returns

R_1 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* **THEN**
TEST: When a call to *sem_init*() completes successfully, the semaphore *sem* is initialized.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion sem_init in 11.2.1.2

R_2 **IF** *PCTS_sem_init* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IF *PCTS_GAP_sem_init* **THEN**
TEST: When a call to *sem_init*() completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and does not initialize the semaphore.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: All assertions in 11.2.1.4

11.2.1.4 Errors

8 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* and {SEM_VALUE_MAX}< {UINT_MAX} **THEN**
TEST: A call to *sem_init(sem, pshared, val)*, when *val* exceeds {SEM_VALUE_MAX}, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* **THEN**
TEST: When a resource required to initialize the semaphore has been exhausted, a call to *sem_init*() returns a value of -1 and sets *errno* to [ENOSPC].
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

10 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* and *PCTS_SEM_IS_FD* and ({OPEN_MAX} < *PCTS_OPEN_MAX)* **THEN**
SETUP: Open {OPEN_MAX} files. Try to open a semaphore.
TEST: When a resource required to initialize the semaphore has been exhausted, a call to *sem_init*() returns a value of -1 and sets *errno* to [ENOSPC].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* and *PCTS_SEM_IS_FD* and ({OPEN_MAX} >= *PCTS_OPEN_MAX)* **THEN**
SETUP: Open *PCTS_OPEN_MAX* files. Try to open a semaphore.
TEST: When a resource required to initialize the semaphore has been exhausted, a call to *sem_init*() returns a value of -1 and sets *errno* to [ENOSPC].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

12 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* and {SEN_NSEMS_MAX}< *PCTS_SEM_NSEMS_MAX* **THEN**
TEST: A call to *sem_init*(), after {SEM_NSEMS_MAX} semaphores have been created, returns a value of -1 and sets *errno* to [ENOSPC].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* and {SEN_NSEMS_MAX}>= *PCTS_SEM_NSEMS_MAX* **THEN**
TEST: A call to *sem_init*(), after {PCTS_SEM_NSEMS_MAX} semaphores have been created, returns a value of -1 and sets *errno* to [ENOSPC].

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

14 **IF** not *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* **THEN**
TEST: A call to *sem_init*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_RAP_sem_init* **THEN**
TEST: A call to *sem_init*(), when the process lacks the appropriate privileges to initialize a semaphore, returns a value of -1 and sets *errno* to [EPERM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

11.2.2 Destroy an Unnamed Semaphore

Function: *sem_destroy*().

11.2.2.1 Synopsis

1
M_GA_stdC_proto_decl(int; sem_destroy; sem_t *sem, semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sem_destroy: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(sem_destroy; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sem_destroy: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; sem_destroy; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sem_destroy: PASS, NO_OPTION

4
M_GA_macro_args (sem_destroy; semaphore.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sem_destroy: PASS, NO_OPTION

11.2.2.2 Description

sem_destroy
IF *PCTS_sem_destroy* **THEN**
TEST: A call *sem_destroy(sem)* destroys the unnamed semaphore *sem* and returns 0.
ELSE *NO_OPTION*
Conformance for sem_destroy: PASS, NO_OPTION

D_1 **IF** *PCTS_sem_destroy* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect of calling *sem_destroy*() with a named semaphore, does so in 11.2.2.2.
ELSE *NO_OPTION*
Conformance for sem_destroy: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D_2 **IF** *PCTS_sem_destroy* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect of using the semaphore *sem* after it is destroyed by a call to *sem_destroy(),* does so in 11.2.2.2.
ELSE *NO_OPTION*
Conformance for sem_destroy: PASS, NO_OPTION

D_3 **IF** *PCTS_sem_destroy* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect of destroying a semaphore upon, which other processes are currently blocked, does so in 11.2.2.2.
ELSE *NO_OPTION*
Conformance for sem_destroy: PASS, NO_OPTION

D_4 **IF** *PCTS_sem_destroy* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sem_destroy*() function, does so in 11.2.2.2.
ELSE *NO_OPTION*
Conformance for sem_destroy: PASS, NO_OPTION

11.2.2.3 Returns

R_1 **IF** *PCTS_sem_destroy* **THEN**
TEST: When a call to *sem_destroy*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion **sem_destroy** in 11.2.2.2.

R_2 **IF** *PCTS_sem_destroy* **THEN**
TEST: When a call to *sem_destroy*() completes unsuccessfully, interface returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 11.2.2.4.

11.2.2.4 Errors

5 **IF** *PCTS_sem_destroy* **THEN**
TEST: The call *sem_destroy(sem),* when the argument *sem* is not a valid semaphore, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid semaphore, or indicates that there is no way to generate an invalid semaphore on the system.
ELSE *NO_OPTION*
Conformance for sem_destroy: PASS, NO_OPTION

6 **IF** not *PCTS_sem_destroy* **THEN**
TEST: A call to *sem_destroy*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*
Conformance for sem_destroy: PASS, NO_OPTION

7 **IF** *PCTS_sem_destroy* and *PCTS_SEM_EBUSY* **THEN**
TEST: A call to *sem_destroy(sem),* when there are currently processes blocked on the semaphore, returns a value of -1 and sets *errno* to [EBUSY].
ELSE *NO_OPTION*
Conformance for sem_destroy: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11.2.3 Initialize/Open a Named Semaphore

Function: *sem_open*().

11.2.3.1 Synopsis

1

M_GA_stdC_proto_decl(sem_t *; sem_open; const char *name, int oflag, ...; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3
Conformance for sem_open: PASS[1, 2], NO_OPTION

2

M_GA_commonC_result_decl(sem_t*; sem_open; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3
Conformance for sem_open: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(sem_t*; sem_open; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4
Conformance for sem_open: PASS, NO_OPTION

4

M_GA_macro_args (sem_open; semaphore.h;;;)
SEE: Assertion GA_macro_args in 2.7.3
Conformance for sem_open: PASS, NO_OPTION

11.2.3.2 Description

M_GA_semOpenMaxFD(PCTS_SEM_OPEN_FD; sem_open; PCTS_sem_open)
SEE: Assertion GA_semOpenMaxFD in 11.1
Conformance for sem_open: PASS/OpenMaxSems, PCTSOpenMaxSems]

M_GA_semPCTSOpenMaxFD(PCTS_SEM_OPEN_FD; sem_open; PCTS_sem_open)
SEE: Assertion GA_semPCTSOpenMaxFD in 11.1
Conformance for sem_open: PASS/OpenMaxSems, PCTSOpenMaxSems]

sem_open

FOR: *sem_wait*(), *sem_trywait*(), *sem_post*(), and *sem_close*()
IF *PCTS_sem_open* **THEN**
IF *PCTS_function* **THEN**
TEST: A call *sem_open(name, oflag, ...)* establishes a connection between a named semaphore, *name*, and the calling process, and returns the address of the named semaphore that can be used in subsequent calls to *function*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_sem_open* **THEN**
SETUP: Include the header `<semaphore.h>`.
TEST: The constants O_CREAT and O_EXCL are defined and are bitwise distinct.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

6 **FOR:** *sem_close*(), *_exit*(), and *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), and *execvp*()
IF *PCTS_sem_open* **THEN**
SETUP: Create a semaphore using *sem_open*().

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: The semaphore remains usable by the calling process until it is closed by a successful call to *function*().

TR: Test using *sem_wait*(), *sem_trywait*(), and *sem_post*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

7 **IF** *PCTS_sem_open* **THEN**

SETUP: Open a semaphore by calling *sem_open(name, oflag, ...)*, where the flag bit O_CREAT is set in *oflag* and the named semaphore *name* does not already exist.

TEST: The named semaphore is created.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

8 **IF** *PCTS_sem_open* **THEN**

SETUP: The call *sem_open(name, oflag, mode, value)* creates a semaphore with an initial value of *value.*

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

9 **IF** *PCTS_sem_open* **THEN**

TEST: Valid initial values for semaphores are unsigned integers with values between 0 and {SEM_VALUE_MAX}, inclusive.

TR: Test values of 0, {SEM_VALUE_MAX}, and {SEM_VALUE_MAX} + 1, if it is less than {UINT_MAX}.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

10 **IF** *PCTS_sem_open* **THEN**

TEST: The user ID of the semaphore created by *sem_open*() is the effective user ID of the process.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

11 **IF** *PCTS_sem_open* **THEN**

TEST: The group ID of the semaphore created by *sem_open*() is a system default group ID, or the effective group ID, of the process.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

12 **IF** *PCTS_sem_open* **THEN**

TEST: The permission bits of the semaphore created by *sem_open(name, oflag, mode, value)* are set to the value of the *mode* argument, except those set in the file mode creation mask, of the process.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

D_1 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the effect of calling *sem_open (name, oflag, mode, value)* with bits specified in *mode,* other than the file permission bits, does so in 11.2.3.2.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

13 **IF** *PCTS_sem_open* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SETUP: Create a new named semaphore with the call *sem_open(name, oflag, ...)* and the O_CREAT flag set in *oflag.*

TEST: Other processes can connect to the semaphore by calling *sem_open*() with the same value of *name.*

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

14 **IF** *PCTS_sem_open* **THEN**

TEST: When the semaphore *name* exists, the call *sem_open (name, oflag, ...)* with the O_CREAT and O_EXCL flags set in *oflag,* fails.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

15 **IF** *PCTS_sem_open* **THEN**

TEST: The check for the existence of the semaphore in a call to *sem_open( )* with the O_EXCL and O_CREAT flags set, and the creation of the semaphore if it does not exist, are atomic with respect to other processes executing *sem_open*() with O_EXCL and O_CREAT set.

NOTE: There is no known reliable test method for this assertion.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_TEST, NO_OPTION

D_2 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the effect of calling *sem_open*(), with O_EXCL set and O_CREAT not set, does so in 11.2.3.2.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

D_3 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the effect of calling *sem_open*() with oflag other than O_EXCL and O_CREAT specified in the *oflag* parameter, does so in 11.2.3.2.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

D_4 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether *name* appears in the filesystem does so in 11.2.3.2.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

D_5 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether *name* is visible to functions that take pathnames as arguments does so in 11.2.3.2.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

16 ***M_GA_portableFilenames(sem_open)***

SEE: Assertion GA_portableFilenames in 2.2.2.40.

Conformance for sem_open: PASS, NO_OPTION

17 ***M_GA_upperLowerNames(sem_open)***

SEE: Assertion GA_upperLowerNames in 2.2.2.40.

Conformance for sem_open: PASS, NO_OPTION

18 ***M_GA_PRNoTrunc(sem_open)***

SEE: Assertion GA_PRNoTrunc in 2.3.6.

Conformance for sem_open: PASS, NO_OPTION

19 ***M_GA_PRNoTruncError(sem_open)***

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SEE: Assertion GA_PRNoTruncError in 2.3.6
Conformance for sem_open: PASS, NO_OPTION

20 **IF** *PCTS_sem_open* **THEN**
SETUP: Call *sem_open(name, oflag, ...)* where *name* begins with the slash character.
TEST: Processes calling *sem_open*() with the same value of *name* refer to the same semaphore object, as long as that name has not been removed.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_6 **IF** *PCTS_sem_open* **THEN**
TEST: A PCD.1b documents the effect of calling *sem_open(name, oflag, ...),* when *name* does not begin with a slash character, in 11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_7 **IF** *PCTS_sem_open* **THEN**
TEST: A PCD.1b documents the interpretation of slash characters in 11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

21 **IF** *PCTS_sem_open* **THEN**
SETUP: Call *sem_open(name, oflag, ...)* with a single value for *name* and no intervening calls to *sem_unlink*().
TEST: The same semaphore address is returned for each successful call.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_8 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effects of references to copies of semaphores created with *sem_open*() does so in 11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_9 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sem_open*() function does so in 11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

11.2.3.3 Returns

R_1 **IF** *PCTS_sem_open* **THEN**
TEST: When a call to *sem_open* () completes successfully, the address of the semaphore is returned.
ELSE *NO_OPTION*
SEE: Assertion sem_open in 11.2.3.2

11.2.3.4 Errors

22 **IF** *PCTS_sem_open* **THEN**
TEST: A call to *sem_open*() when the named semaphore exists and the permissions specified by *oflag* are denied, or the named semaphore does not exist and permission to create the named semaphore is denied, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

23 **IF** *PCTS_sem_open* **THEN**
TEST: A call to *sem_open*() when O_CREAT and O_EXCL are set and the named semaphore already exists, returns a value of -1 and sets *errno* to [EEXIST].
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

24 **IF** *PCTS_sem_open* **THEN**
TEST: A call to *sem_open*(), when the *sem_open*() operation is interrupted by a signal, returns a value of -1 and sets *errno* to [EINTR]
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

25 **IF** *PCTS_sem_open* **THEN**
TEST: A call to *sem_open*(), when the *sem_open*() operation is not supported for the given name, returns a value of -1 and sets *errno* to [EINVAL]
NOTE: A subroutine is recommended that either returns a name for which *sem_open*() is not supported, or indicates that there is no way to generate *sem_open*() on the system.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_10 **IF** *PCTS_sem_open* **THEN**
TEST: The PCD.1b documents under what circumstances [EINVAL] may be returned in 11.2.3.4.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

26 **IF** *PCTS_sem_open* **THEN**
TEST: A call to *sem_open(name, oflag, ...),* when O_CREAT is specified in *oflag* and *value* is greater than {SEM_VALUE_MAX}, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

27 **IF** *PCTS_sem_open* **THEN**
TEST: A call to *sem_open*(), when too many semaphore descriptors are currently in use by this process, returns a value of -1 and sets *errno* to [EMFILE].
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

28 **IF** *PCTS_sem_open* **THEN**
IF *PCTS_SEM_IS_FD* and ({OPEN_MAX}< *PCTS_OPEN_MAX)* **THEN**
TEST: A call to *sem_open*(), when too many file descriptors are currently in use by this process, returns a value of -1 and sets *errno* to [EMFILE]
TR: Open {OPEN_MAX} files. Try to open a semaphore.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_TEST_SUPPORT, NO_OPTION

29 **IF** *PCTS_sem_open* **THEN**
IF {PATH_MAX}<= *PCTS_PATH_MAX)* **THEN**
TEST: A call to *sem_open*(), when the length of the *name* string exceeds {PATH_MAX}, returns a value of -1 and sets *errno* to [ENAMETOOLONG]
TR: Open {OPEN_MAX} files. Try to open a semaphore.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30 **IF** *PCTS_sem_open* **THEN**
IF {NAME_MAX}<= *PCTS_NAME_MAX)* and {_POSIX_NO_TRUNC} **THEN**
TEST: A call to *sem_open*(), when a pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC) is in effect, returns a value of -1 and sets *errno* to [ENAMETOOLONG].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_TEST_SUPPORT, NO_OPTION

31 **IF** *PCTS_sem_open* **THEN**
TEST: A call to *sem_open*(), when too many semaphores are currently open in the system, returns a value of -1 and sets *errno* to [ENFILE].
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

32 **IF** *PCTS_sem_open* **THEN**
TEST: A call to *sem_open*(), when O_CREAT is not set and the named semaphore does not exist, returns a value of -1 and sets *errno* to [ENOENT].
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

33 **IF** *PCTS_sem_open* **THEN**
TEST: A call to *sem_open*(), when there is insufficient space for the creation of the new named semaphore, returns a value of -1 and sets *errno* to [ENOSPC].
NOTE: There is no known reliable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_TEST, NO_OPTION

34 **IF** not *PCTS_sem_open* **THEN**
TEST: A call to *sem_open*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

11.2.4 Close a Named Semaphore

Function: *sem_close*().

11.2.4.1 Synopsis

1

M_GA_stdC_proto_decl(int; sem_close; sem_t *sem; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sem_close: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sem_close; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sem_close: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sem_close; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sem_close: PASS, NO_OPTION

4

M_GA_macro_args (sem_close; semaphore.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sem_close: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11.2.4.2 Description

sem_close

IF *PCTS_sem_close* **THEN**
IF: *PCTS_sem_open* and {SEN_NSEMS_MAX} < *PCTS_SEM_NSEMS_MAX* **THEN**
TEST: The *sem_close*() function makes any system resources allocated by the system available for reuse by a subsequent *sem_open*() call in this process and returns 0.
TR: Open {SEM_NSEMS_MAX} semaphores, close one, then open another.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_sem_close* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effects of calling *sem_close*() for an unnamed semaphore, does so in 11.2.4.2.
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

D_2 **IF** *PCTS_sem_close* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effects of subsequent use of the semaphore *sem,* does so in 11.2.4.2.
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

5 **IF** *PCTS_sem_close* **THEN**
TEST: The *sem_close*() has no effect on the state of the semaphore, if the semaphore has not been removed with a successful call to *sem_unlink*().
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

6 **IF** *PCTS_sem_close* **THEN**
IF: *PCTS_sem_open* *PCTS_sem_unlink* **THEN**
SETUP: Call *sem_open*() with O_CREAT; then call *sem_unlink*() successfully on the same semaphore.
TEST: When all processes that have opened the semaphore then close it, the semaphore is no longer accessible.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_TEST_SUPPORT, NO_OPTION

D_3 **IF** *PCTS_sem_close* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sem_close*() function, does so in 11.2.4.2.
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

11.2.4.3 Returns

R_1 **IF** *PCTS_sem_close* **THEN**
TEST: When a call to *sem_close*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion sem_close in 11.2.4.2.

R_2 **IF** *PCTS_sem_close* **THEN**
TEST: When a call to *sem_close*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ELSE *NO_OPTION*
SEE: Assertion sem_close in 11.2.4.4.

11.2.4.4 Errors

7 **IF** *PCTS_sem_close* **THEN**
TEST: A call to *sem_close*(), when the *sem* argument is not a valid semaphore descriptor, returns a value of -1 and sets *errno* to {EINVAL}.
NOTE: A subroutine is recommended that either returns an invalid semaphore descriptor, or indicates that there is no way to generate an invalid semaphore descriptor on the system.
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

8 **IF** not *PCTS_sem_close* **THEN**
TEST: A call to *sem_close*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

11.2.5 Remove a Named Semaphore

Function: *sem_unlink*().

11.2.5.1 Synopsis

1
M_GA_stdC_proto_decl(int; sem_unlink; const char *name; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sem_unlink: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(sem_unlink; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sem_unlink: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; sem_unlink; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sem_unlink: PASS, NO_OPTION

4
M_GA_macro_args (sem_unlink; semaphore.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sem_unlink: PASS, NO_OPTION

11.2.5.2 Description

sem_unlink
IF *PCTS_sem_unlink* **THEN**
TEST: The call *sem_unlink(name)* removes the semaphore named by the string *name* and returns the value zero.
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5 **IF** *PCTS_sem_unlink* **THEN**
TEST: When the semaphore named by name is currently referenced by other processes, then *sem_unlink(name)* has no effect on the state of the semaphore.
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

6 **FOR:** *sem_close*(), *_exit*(), *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), and *execvp*()
IF *PCTS_sem_unlink* **THEN**
TEST: When one or more processes have the semaphore open when *sem_unlink*()is called, destruction of the semaphore is postponed until all references to the semaphore have been destroyed by a call to *function*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

7 **IF** *PCTS_sem_unlink* **THEN**
IF *PCTS_sem_open*
TEST: Calls to *sem_open*() to recreate or reconnect to the semaphore refer to a new semaphore after *sem_unlink*() is called.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_sem_unlink* **THEN**
TEST: When any process references *name*, the call *sem_unlink(name)* returns immediately.
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

D_1 **IF** *PCTS_sem_unlink* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sem_unlink*() function does so in 11.2.5.2.
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

11.2.5.3 Returns

R_1 **IF** *PCTS_sem_unlink* **THEN**
TEST: When a call to *sem_unlink*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion sem_unlink in 11.2.5.2.

R_2 **IF** *PCTS_sem_unlink* **THEN**
TEST: When a call to *sem_unlink*() completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and does not change the semaphore.
ELSE *NO_OPTION*
SEE: All assertions in 11.2.5.4

11.2.5.4 Errors

9 **IF** *PCTS_sem_unlink* **THEN**
TEST: A call to *sem_unlink*(), when permission is denied to unlink the named semaphore, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10 **IF** *PCTS_sem_unlink* **THEN**
IF *{_posix_no_truck} AND {name_max}<=* ***PCTS_NAME_MAX*** **THEN**
TEST: A call to *sem_unlink*(), when the length of the *name* string exceeds {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect, returns a value of -1 and sets *errno* to [ENAMETOOLONG].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_sem_unlink* **THEN**
TEST: A calls to *sem_unlink*(), when the named semaphore does not exist, returns a value of -1 and sets *errno* to [ENOENT].
NOTE: A subroutine is recommended that either returns the name of a semaphore that does not exist, or indicates that there is no way to generate the name of a semaphore that does not exist in the system.
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

12 **IF** not *PCTS_sem_unlink* **THEN**
TEST: A calls to *sem_unlink*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

11.2.6 Lock a Semaphore

Functions: *sem_wait*(), *sem_trywait*().

11.2.6.1 Synopsis

1
M_GA_stdC_proto_decl(int; sem_wait; sem_t *sem; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sem_wait: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(sem_wait; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sem_wait: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; sem_wait; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sem_wait: PASS, NO_OPTION

4
M_GA_macro_args (sem_wait; semaphore.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sem_wait: PASS, NO_OPTION

5
M_GA_stdC_proto_decl(int; sem_trywait; sem_t *sem; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sem_trywait: PASS[5, 6], NO_OPTION

6
M_GA_commonC_int_result_decl(sem_trywait; sem_t*sem; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sem_trywait: PASS[5, 6], NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7

M_GA_macro_result_decl(int; sem_trywait; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sem_trywait: PASS, NO_OPTION

8

M_GA_macro_args (sem_trywait; semaphore.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sem_trywait: PASS, NO_OPTION

11.2.6.2 Description

sem_wait

FOR: *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When the call *sem_wait(sem)* locks the semaphore *sem*, with the semaphore lock operation, it returns 0.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

sem_trywait

FOR: *sem_init*() and *sem_open*()
IF *PCTS_sem_trywait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When the call *sem_trywait(sem)* locks the semaphore *sem*, with the semaphore lock operation, it returns the value zero.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When the semaphore value is currently zero, and the call to *sem_wait(sem)* is not interrupted by a signal, the call returns when the semaphore referenced by *sem* is locked.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

10 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_trywait* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: When the semaphore value is currently positive, and the call to *sem_wait(sem)* is not interrupted by a signal, the call returns 0 when the semaphore referenced by *sem* is locked.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_trywait* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: When the semaphore value is currently zero, the call to *sem_trywait*() does not lock the semaphore referenced by *sem.*

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

12 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_wait* **THEN**

IF *PCTS_function* and *sem_post*()**THEN**

SETUP: Create a semaphore using *function*().

TEST: After a successful *sem_wait*() call, the semaphore is locked, and remains locked, until the *sem_post*() function is executed and returns successfully.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_trywait* **THEN**
IF *PCTS_function* and *sem_post*() **THEN**
SETUP: Create a semaphore using *function*().
TEST: After a successful call to *sem_trywait*(), the semaphore is locked, and remains locked, until the *sem_post* function is executed and returns successfully.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

14 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: The *sem_wait*() function is interruptible by the delivery of a signal.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_sem_wait* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sem_wait*() function does so in 11.2.6.2.
ELSE *NO_OPTION*
Conformance for sem_wait: PASS, NO_OPTION

D_2 **IF** *PCTS_sem_trywait* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sem_trywait*() function does so in 11.2.6.2.
ELSE *NO_OPTION*
Conformance for sem_trywait: PASS, NO_OPTION

11.2.6.3 Returns

15 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: The *sem_wait(sem)* function returns 0 if the calling process successfully performed the semaphore lock operation on the semaphore designated by *sem.*

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_trywait* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: The *sem_trywait(sem)* function returns 0 if the calling process successfully performed the semaphore lock operation on the semaphore designated by *sem.*

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

R_1 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_wait* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: When a call to *sem_wait(sem)* completes successfully, the interface returns a value of 0 and the semaphore designated by *sem* is locked by the semaphore lock operation.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion sem_wait in 11.2.6.2.

R_2 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_trywait* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: When a call to *sem_trywait(sem)* completes successfully, the interface returns a value of 0 and the semaphore designated by *sem* is locked by the semaphore lock operation.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion sem_trywait in 11.2.6.2.

R_3 FOR: *sem_init*() and *sem_open*()

IF *PCTS_sem_wait* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: When a call to *sem_wait(sem)* completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and does not change the state of the semaphore.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: All assertions in 11.2.6.4 controlled by *PCTS_sem_wait.*

R_4 FOR: *sem_init*() and *sem_open*()

IF *PCTS_sem_trywait* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: When a call to *sem_trywait*() completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and does not change the state of the semaphore.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: All assertions in 11.2.6.4 controlled by *PCTS_sem_trywait.*

11.2.6.4 Errors

17 FOR: *sem_init*() and *sem_open*()

IF *PCTS_sem_wait* **THEN**

IF *PCTS_function* **THEN**

TEST: A call to *sem_wait*(), when the semaphore is already locked, so that it cannot be immediately locked by the *sem_trywait*() operation, returns a value of -1 and sets *errno* to [EAGAIN].

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_trywait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: A call to *sem_trywait*(), when the semaphore is already locked, so that it cannot be immediately locked by the *sem_trywait*() operation, returns a value of -1 and sets *errno* to [EAGAIN].
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

19 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: A call to *sem_wait*(), when the *sem* argument does not refer to a valid semaphore, returns a value of -1 and sets *errno* to [EINVAL].
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: A subroutine is recommended that either returns an invalid semaphore, or indicates that there is no way to generate an invalid semaphore on the system.
The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

20 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_trywait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: A call to *sem_trywait(sem)*, when the *sem* argument does not refer to a valid semaphore, returns a value of -1 and sets *errno* to [EINVAL].
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: A subroutine is recommended that either returns an invalid semaphore, or indicates that there is no way to generate an invalid semaphore on the system.
The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: A call to *sem_wait*(), interrupted by a signal, returns a value of -1 and sets *errno* to [EINTR].
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

22 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_trywait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: A call to *sem_trywait*(), when interrupted by a signal, returns a value of -1 and sets *errno* to [EINVAL].
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

23 **IF** not *PCTS_sem_wait* **THEN**
TEST: A call to *sem_wait*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sem_wait: PASS, NO_OPTION

24 **IF** not *PCTS_sem_trywait* **THEN**
TEST: A call to *sem_trywait*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sem_trywait: PASS, NO_OPTION

25 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: A call to *sem_wait*(), when a deadlock condition is detected, returns a value of -1 and sets *errno* to [EDEADLK].
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

26 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_trywait* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: A call to *sem_trywait*(), when a deadlock condition is detected, returns a value of -1 and sets *errno* to [EDEADLK].

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

11.2.7 Unlock a Semaphore

Function: *sem_post*().

11.2.7.1 Synopsis

1

M_GA_stdC_proto_decl(int; sem_post; sem_t *sem, semaphore.h;;;)

SEE: Assertion GA_stdC_proto_decl in 2.7.3.

Conformance for sem_post: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sem_post; semaphore.h;;;)

SEE: Assertion GA_commonC_int_result_decl in 2.7.3.

Conformance for sem_post: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sem_post; semaphore.h;;;)

SEE: Assertion GA_macro_result_decl in 1.3.4.

Conformance for sem_post: PASS, NO_OPTION

4

M_GA_macro_args (sem_post; semaphore.h;;;)

SEE: Assertion GA_macro_args in 2.7.3.

Conformance for sem_post: PASS, NO_OPTION

11.2.7.2 Description

sem_post

FOR: *sem_init*() and *sem_open*()

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: A successful call to *sem_sempost*() unlocks the semaphore referenced by *sem* by performing the semaphore unlock operation on that semaphore, and returns the value 0.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: When the semaphore value resulting from *sem_post*() is positive, then the semaphore value is incremented.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_post* **THEN**

IF *PCTS_sem_wait* **THEN**

SETUP: Create a semaphore using *function*(). Also, create multiple processes and have them block waiting on the semaphore.

TEST: When the value of the semaphore resulting from *sem_post*() is 0, then one of the processes blocked waiting for the semaphore returns successfully from its call to *sem_wait*().

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

R_1 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* and {_POSIX_PRIORITY_SCHEDULING} **THEN**

SETUP: Create a semaphore using *function*().

TEST: When the value of the semaphore resulting from *sem_post*() is 0, the process to be unblocked is chosen in a manner appropriate to the scheduling policies and parameters in effect for the blocked processes.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SEE: Assertion 8 in 11.2.7.2.

7 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_post* **THEN**
IF *PCTS_function* and {_POSIX_PRIORITY_SCHEDULING}**THEN**
SETUP: Create a semaphore using *function*().
TEST: When the value of the semaphore resulting from *sem_post*() is 0, and the scheduler is SCHED_FIFO or SCHED_RR, the highest priority waiting process is unblocked.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

Test for each of SCHED_FIFO and SCHED_RR.
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_post* **THEN**
IF *PCTS_function* and {_POSIX_PRIORITY_SCHEDULING}**THEN**
SETUP: Create a semaphore using *function*().
TEST: When the value of the semaphore resulting from *sem_post*() is 0, and the scheduler is SCHED_FIFO or SCHED_RR, and there is more than one highest priority process blocked waiting for the semaphore, then the highest priority process that has been waiting the longest is unblocked.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

Test for each of SCHED_FIFO and SCHED_RR.
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_sem_post* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the choice of a process to unblock if {_POSIX_PRIORITY_SCHEDULING} is defined does so in 11.2.7.2.
ELSE *NO_OPTION*
Conformance for sem_post: PASS, NO_OPTION

9 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_post* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: The *sem_post*() function is reentrant with respect to signals and may be invoked from a signal-catching function.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 IF *PCTS_sem_post* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *sem_post*() function does so in 11.2.7.2.

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_OPTION

11.2.7.3 Returns

R_2 FOR: *sem_init*() and *sem_open*()

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: When a call to *sem_post*() completes successfully, the interface returns a value of 0.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion sem_post in 11.2.7.2.

R_3 FOR: *sem_init*() and *sem_open*()

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: When a call to *sem_post*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: All assertions in 11.2.7.4.

11.2.7.4 Errors

10 FOR: *sem_init*() and *sem_open*()

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SETUP: Create a semaphore using *function*().
TEST: A call to *sem_post*(), when the *sem* does not refer to a valid semaphore, returns a value of -1 and sets *errno* to [EINVAL].
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: A subroutine is recommended that either returns an invalid semaphore, or indicates that there is no way to generate an invalid semaphore on the system.

The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** not *PCTS_sem_post* **THEN**
TEST: A call to *sem_post*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sem_post: PASS, NO_OPTION

11.2.8 Get the Value of a Semaphore

Function: *sem_getvalue*().

11.2.8.1 Synopsis

1
M_GA_stdC_proto_decl(int; sem_getvalue; sem_t *sem, int *sval; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sem_getvalue: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(sem_getvalue; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sem_getvalue: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; sem_getvalue; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sem_getvalue: PASS, NO_OPTION

4
M_GA_macro_args (sem_getvalue; semaphore.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sem_getvalue: PASS, NO_OPTION

11.2.8.2 Description

sem_getvalue
FOR: *sem_init*() and *sem_open*()
IF *PCTS_sem_getvalue* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: A successful call of *sem_getvalue(sem, sval)* updates the location referenced by the *sval* argument to have the value of the semaphore referenced by *sem* at some unspecified time during the call, and returns the value of 0.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_getvalue: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_getvalue* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: The *sem_getvalue*() function does not affect the state of the semaphore referenced by *sem.*

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_getvalue: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **FOR:** *sem_init*() and *sem_open*()

IF *PCTS_sem_getvalue* **THEN**

IF *PCTS_function* **THEN**

SETUP: Create a semaphore using *function*().

TEST: When *sem* is locked, then the value returned by *sem_getvalue*() is either 0 or a negative number whose absolute value represents the number of processes waiting for the semaphore at some unspecified time during the call.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_getvalue: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_sem_getvalue* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *sem_getvalue*() function does so in 11.2.8.2.

ELSE *NO_OPTION*

Conformance for sem_getvalue: PASS, NO_OPTION

11.2.8.3 Returns

R_1 **FOR:** *sem_init*() and *sem_open*()

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IF *PCTS_sem_getvalue* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When a call to *sem_getvalue*() completes successfully, the interface returns a value of 0.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion sem_getvalue in 11.2.8.2.

R_2 FOR: *sem_init*() and *sem_open*()
IF *PCTS_sem_getvalue* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When a call to *sem_getvalue*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: All assertions in 11.2.8.4

7 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_getvalue* **THEN**
IF *PCTS_function* and *PCTS_SEM_INVALID* **THEN**
SETUP: Create a semaphore using *function*().
TEST: A call to *sem_getvalue*(), when the *sem* argument does not refer to a valid semaphore, returns a value of -1 and sets *errno* to [EINVAL]
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is no way to get appropriate privilege to call *sem_init*().
NOTE: A subroutine is recommended that either returns an invalid semaphore or indicates that there is no way to generate an invalid semaphore on the system.

The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_getvalue: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **IF** not *PCTS_sem_getvalue* **THEN**
TEST: A call to *sem_getvalue*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sem_getvalue: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

--`,,`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Section 12: Memory Management

1 **TEST:** The page size, in bytes, is the value of the configurable system variable [PAGESIZE].
Conformance for mem_intro: PASS

D_1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the restriction of the size and alignment of range lockings and mappings to be on page-size boundaries, does so in 12.
ELSE *NO_OPTION*
Conformance for mem_intro: PASS, NO_OPTION

D_2 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents 1B page size, meaning no restrictions on the size or alignment of range lockings and mappings, does so in 12.
ELSE *NO_OPTION*
Conformance for mem_intro: PASS, NO_OPTION

D_3 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether locking memory guarantees fixed translation between virtual addresses (as seen by the process) and physical addresses, does so in 12.
ELSE *NO_OPTION*
Conformance for mem_intro: PASS, NO_OPTION

R_1 **IF** *PCTS_mlockall* or *PCTS_mlock* **THEN**
TEST: Per-process memory locks are not inherited across a *fork*( ).
ELSE *NO_TEST_SUPPORT*
SEE: Assertion 2 in 3.1.1.2.

R_2 **IF** *PCTS_mlockall* or *PCTS_mlock* **THEN**
TEST: All memory locks owned by a process are unlocked upon *exec* or process termination.
ELSE *NO_TEST_SUPPORT*
SEE: Assertion mlock in 12.1.2.2.

R_3 **IF** *PCTS_munlock* **THEN**
TEST: Unmapping of an address range removes any memory locks established on that address range by this process.
ELSE *NO_TEST_SUPPORT*
SEE: Assertion munlock_remove_maps in 12.2.2.2.

2 **IF** *PCTS_mmap* **THEN**
TEST: Once a file is "mapped" into a process address space, the data can be manipulated as memory.
ELSE *NO_TEST_SUPPORT*
Conformance for mem_intro: PASS, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3 **IF** *PCTS_mmap* **THEN**
TEST: When more than one process maps a file, its contents are shared among them.
ELSE *NO_TEST_SUPPORT*
Conformance for mem_intro: PASS, NO_TEST_SUPPORT

4 **IF** *PCTS_mmap* **THEN**
TEST: When the mappings allow shared write access, then data written into the memory object through the address space of one process appear in the address spaces of all processes that similarly map the same portion of the memory object.
ELSE *NO_TEST_SUPPORT*
Conformance for mem_intro: PASS, NO_TEST_SUPPORT

R_4 **IF** *PCTS_shm_unlink* **THEN**
TEST: *unlink* ( ) of a mapped file or *shm-unlink* ( ) of a shared memory object, while causing the removal of the name, does not unmap any mappings while causing the removal of the name, and does not unmap any mappings established for the object. Once the name has been removed, the contents of the memory object are preserved as long as a process has the memory object open, or has some area of the memory object mapped.
ELSE *NO_TEST_SUPPORT*
SEE: All assertions in 12.3.2.2.

R_5 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**
IF *PCTS__mmap* **THEN**
TEST: References to whole pages within the mapping but beyond the current length of an object result in a SIGBUS signal.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion mmap_ SIGBUS in 12.1.2.2.

D_4 **IF** {_POSIX_MEMORY_PROTECTION} and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of references to memory within the mapping but beyond the current length of an object, does so in 12.
ELSE *NO_OPTION*
Conformance for mem_intro: PASS, NO_OPTION

5 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**
IF: *PCTS_mmap* **THEN**
SETUP: Create a mapped memory object using *mmap* ( ).
TEST: The size of a memory object is unaffected by access beyond the end of the object.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mem_intro: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**
IF: *PCTS_mmap* **THEN**
SETUP: Create a mapped memory object without write access using *mmap* ( ).
TEST: Write attempts to memory mapped without write access result in a SIGSEGV signal.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mem_intro: PASS, NO_TEST_SUPPORT, NO_OPTION

7 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**
IF: *PCTS_mmap* **THEN**
SETUP: Create a mapped memory object with PROT_NONE using *mmap* ( ).
TEST: Any access to memory mapped PROT_NONE results in a SIGSEGV signal.
ELSE *NO_TEST_SUPPORT*

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ELSE *NO_OPTION*
Conformance for mem_intro: PASS, NO_TEST_SUPPORT, NO_OPTION

R_6 IF {_POSIX_MEMORY_PROTECTION} **THEN**
TEST: References to unmapped addresses result in a SIGSEGV signal.
ELSE *NO_OPTION*
SEE: Assertion munmap_ SIGSEV in 12.2.2.2.

D_5 IF not {_POSIX_MEMORY_PROTECTION} and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect of references to unmapped addresses, does so in 12.
ELSE *NO_OPTION*
Conformance for mem_intro: PASS, NO_OPTION

12.1 Memory Locking Functions

12.1.1 Lock/Unlock the Address Space of a Process

Functions: *mlockall*( ), *munlockall* ().

12.1.1.1 Synopsis

1

M_GA_stdC_proto_decl(int; mlockall; int; int flags; sys/mman .h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for mlockall: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(mlockall; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for mlockall: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; mlockall; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for mlockall: PASS, NO_OPTION

4

M_GA_macro_args (mlockall; sys/mman.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mlockall: PASS, NO_OPTION

5

M_GA_stdC_proto_decl(int; munlockall; sys/mman .h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for munlockall: PASS[5, 6], NO_OPTION

6

M_GA_commonC_int_result_decl(munlockall; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for munlockall: PASS[5, 6], NO_OPTION

7

M_GA_macro_result_decl(int; munlockall; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for munlockall: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8

M_GA_macro_args (munlockall; sys/mman.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for munlockall: PASS, NO_OPTION

12.1.1.2 Description

mlockall
FOR: *exel*( ), *execv*( ), *execle*( ), *execve*( ), *execle*( ), and *execve*( )
IF *PCTS_mlockall* **THEN**
IF: *PCTS_GAP_mlockall* **THEN**
TEST: A successful call to the function *mlockall*() returns 0 and makes all of the pages mapped by the address space of a process memory resident until unlocked, or until the process exits or executes a successful call to *function*( ).
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*( ) with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function*
TR: Try both *exec* and *exit*( ).
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_mlockall* **THEN**
SETUP: Include the header `<sys/mman.h>`
TEST: The constants MCL_CURRENT and MCL_FUTURE are defined and are bitwise distinct.
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_OPTION

10 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_GAP_mlockall* **THEN**
TEST: When the flag MCL_CURRENT is set, the call *mlockall (flags)* locks all of the pages currently mapped into the address space of the process.
TR: Test for at least two disjoint sets of pages.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_GAP_mlockall* **THEN**
TEST: When the flag MCL_FUTURE is set, the call *mlockall (flags)* locks all of the pages that become mapped into the address space of the process in the future, when those mappings are established.
TR: Test for at least two disjoint sets of pages.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_mlockall* **THEN**
TEST: The PCD.1b that documents the behavior if MCL_FUTURE is specified in 12.1.1.2, as well as
1. The automatic locking of future mappings eventually causes the amounts of locked memory to exceed the amount of available physical memory
2. the automatic locking of future mappings eventually causes the amount of locked memory to exceed any other implementation-defined limit,
3. the manner in which the implementation informs the application of these situations.
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for mlockall: PASS, NO_OPTION

munlockall

IF *PCTS_munlockall* **THEN**
TEST: A successful call to *munlockall*() unlocks all currently mapped pages of the address space of the process, with respect to the process's address space, and returns 0.
TR: Test for at least two disjoint sets of pages.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for munlockall: PASS, NO_OPTION

12 **IF** *PCTS_munlockall* **THEN**
IF: *PCTS_GAP_mlockall* **THEN**
TEST: Any pages that become mapped into the address space of the process after a call to *munlockall*() are not locked, unless there is an intervening call to *mlockall*() specifying MCL_FUTURE, or a subsequent call to *mlockall*() specifying MCL_CURRENT.
TR: Test for both MCL_FUTURE and MCL_CURRENT.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_munlockall* **THEN**
TEST: When pages mapped into the address space of the process are also mapped into the address spaces of other processes, and are locked by those processes, the locks established by the other processes are unaffected by a call by this process to *munlockall*().
TR: Test for at least two other processes.
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_OPTION

14 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_GAP_munlockall* **THEN**
TEST: After a successful call to *mlockall*( ) that specifies MCL_CURRENT, all currently mapped pages of the process's address space are memory resident and locked.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_mlockall* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the memory residency of unlocked pages, does so in 12.1.1.2.
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_OPTION

15 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_RAP_mlockall* **THEN**
TEST: Appropriate privilege is required to lock process memory with *mlockall*( ).
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

D_3 **IF** *PCTS_mlockall* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mlockall*( ) function, does so in 12.1.1.2.
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D_4 IF *PCTS_munlockall* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *munlockall*( ) function, does so in 12.1.1.2.
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_OPTION

12.1.1.3 Returns

R_1 IF *PCTS_mlockall* **THEN**
TEST: When a call to *mlockall*() completes successfully, the interface returns a value of 0.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion mlockall in 12.1.1.2

R_2 IF *PCTS_mlockall* **THEN**
TEST: When a call to *mlockall*() completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and no additional memory is locked.
ELSE *NO_OPTION*
SEE: All assertions in 12.1.1.4 controlled by a *PCTS_mlockall*

D_5 IF *PCTS_mlockall* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect of failure of *mlockall*() on previously existing locks in the address space does so in 12.1.1.3.

R_3 IF *PCTS_mlockall* and a PCD.1b documents the following *PCTS_munlockall* **THEN**
TEST: The interface *munlockall*() returns a value of 0
ELSE *NO_OPTION*
SEE: Assertion munlockall in 12.1.1.2

12.1.1.4 Errors

16 **IF** not *PCTS_mlockall* **THEN:**
TEST: A call to *mlockall*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_OPTION

17 **IF** not *PCTS_munlockall* **THEN:**
TEST: A call to *munlockall*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for munlockall: PASS, NO_OPTION

18 **IF** *PCTS_mlockall* **THEN:**
IF *PCTS_GAP_mlockall* **THEN:**
TEST: A call to *mlockall*(),when some or all of the memory identified by the operation could not be locked, returns a value of -1 and sets *errno* to [EAGAIN].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_mlockall* **THEN:**
IF *PCTS_GAP_mlockall* **THEN:**
TEST: A call to *mlockall*(),when the *flags* argument is 0, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

20 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_GAP_mlockall* **THEN:**
TEST: A call to *mlockall*(),when *flags* includes unimplemented flags, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns a *flags* argument that includes unimplemented flags, or indicates that there is no way to generate a *flags* argument that includes unimplemented flags on the system.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_GAP_mlockall*
PCTS_DETECT_LOCKABLE_MEMORY_LIMITS_mlockall **THEN:**
TEST: A call to *mlockall*(),when locking all of the pages currently mapped into the address space of the process would exceed an implementation-defined limit on the amount of memory that the process may lock, returns a value of -1 and sets *errno* to [ENOMEM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

D_6 **IF** *PCTS_mlockall* **THEN**
TEST: The PCD.1b documents the maximum amount of memory that process may lock in 12.1.1.4.
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_OPTION

22 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_RAP_mlockall PCTS_DETECT_NO_AP* **THEN:**
TEST: A call to *mlockall*(),when the calling process does not have the appropriate privilege to perform the requested operation, returns a value of -1 and sets *errno* to [EPERM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

12.1.2 Lock/Unlock a Range of Process Address Space

Functions: *mlock*(), *munlock*().

12.1.2.1 Synopsis

1

M_GA_stdC_proto_decl(int; mlock; const void *addr, size_tlen; sys/mman.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for mlock: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(mlock; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for mlock: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; mlock; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sem_init: PASS, NO_OPTION

4

M_GA_macro_args (mlock; sys/mman.h;;;)

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mlock: PASS, NO_OPTION

5

M_GA_stdC_proto_decl(int; munlock; const void *addr, size_tlen; sys/mman.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for munlock: PASS[5, 6], NO_OPTION

6

M_GA_commonC_int_result_decl(munlock; const void *addr, size_tlen; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for munlock: PASS[5, 6], NO_OPTION

7

M_GA_macro_result_decl(int; munlock; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for munlock: PASS, NO_OPTION

8

M_GA_macro_args (munlock; sys/mman.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for munlock: PASS, NO_OPTION

12.1.2.2 Description

mlock **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*() and *execvp*()
IF *PCTS_MLOCK* **THEN**
IF *PCTS_GAP_mlock* **THEN**
TEST: A successful call to the function *mlock* (*addr, len*) returns a value of 0 and causes those whole pages containing any part of the address space of the process, starting at address *addr* and continuing for *len* bytes, to be memory resident until unlocked, or until the process exits, or executes a successful call to *function*().
TR: Test for each of the three conditions.
NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** a *PCTS_mlock* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the argument *addr*, in a call to *mlock (addr, len)*, must be a multiple of the page size {PAGESIZE}, does so in 12.1.2.2.
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

munlock
IF *PCTS_ munlock* **THEN**
IF *PCTS_mlock* and *PCTS_GAP_mlock* **THEN**
TEST: The call *munlock (addr, len)* unlocks those whole pages containing any part of the address space of the process, with respect to the address space of the process, starting at address *addr* and continuing by *len* bytes, regardless of how many times *mlock*() has been called by the process for any of the pages in the specified ranges, and returns 0.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for munlock: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_munlock* documents the following **THEN**
TEST: A PCD.1b that documents the argument *addr*, in a call to *munlock (addr, len),* must be a multiple of the page size {PAGESIZE}, does so in 12.1.2.2.
ELSE *NO_OPTION*
Conformance for munlock: PASS, NO_OPTION

9 **IF** *PCTS_munlock* **THEN**
TEST: When any of the pages in the range specified by a call to *munlock*() are also mapped into the address spaces of other processes, any locks established on those pages by another process are unaffected by the call of this process to *munlock*().
ELSE *NO_OPTION*
Conformance for munlock: PASS, NO_OPTION

10 **IF** *PCTS_munlock* **THEN**
TEST: When any of the pages in the range specified by a call to *munlock*() are also mapped into the address spaces of other processes, any locks established on those pages via the other mappings are unaffected by this call.
ELSE *NO_OPTION*
Conformance for munlock: PASS, NO_OPTION

D_3 **IF** *PCTS_mlock* and PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents memory residency of unlocked pages does so in 12.1.2.2.
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

11 **IF** *PCTS_ mlock* **THEN**
IF *PCTS_RAP_mlock* **THEN**
TEST: Appropriate privilege is required to lock process memory with *mlock*().
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

D_4 **IF** *PCTS_mlock* and PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mlock*() function does so in 12.1.2.2.
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

D_5 **IF** *PCTS_munlock* and PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports *munlock*() does so in 12.1.2.2.
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

12.1.2.3 Returns

R_1 **IF** *PCTS_mlock* **THEN**
TEST: When a call to *mlock*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion mlock in 12.1.2.2.

R_2 **IF** *PCTS_mlock* **THEN**
TEST: When a call to *mlock*() completes successfully, the interface returns a value of -1, sets *errno* to indicate the error, and no change is made to any locks in the address space.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

PCTS_mlock
ELSE *NO_OPTION*
SEE: All assertions in 12.1.2.4 controlled by *PCTS_mlock*.

R_3 **IF** *PCTS_munlock* **THEN**
TEST: When a call to *munlock*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion munlock in 12.1.2.2.

R_4 **IF** *PCTS_munlock* **THEN**
TEST: When a call to *munlock*() completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and no change is made to any locks in the address space.
PCTS_munlock
ELSE *NO_OPTION*
SEE: All assertions in 12.1.2.4 controlled by *PCTS_munlock*.

12.1.2.4 Errors

12 **IF** *PCTS_mlock* **THEN**
IF: *PCTS_GAP_mlock* **THEN**
TEST: A call to *mlock*(), when some or all of the address range specified by the *addr* and *len* arguments does not correspond to valid mapped pages in the address space of the process, returns a value of -1 and sets *errno* to [ENOMEN].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_munlock* **THEN**
TEST: A call to *munlock*(), when some or all of the address range specified by the *addr* and *len* arguments does not correspond to valid mapped pages in the address space of the process, returns a value of -1 and sets *errno* to [ENOMEN].
ELSE *NO_OPTION*
Conformance for munlock: PASS, NO_OPTION

14 **IF** not *PCTS_mlock* **THEN**
TEST: A call to *mlock*(), returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

15 **IF** not *PCTS_munlock* **THEN**
TEST: A call to *munlock*(), returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

16 **IF** *PCTS_mlock* **THEN**
IF: *PCTS_GAP_mlock* **THEN**
TEST: A call to *mlock*(), when some or all of the memory identified by the operation could not be locked, returns a value of -1 and sets *errno* to [EAGAIN].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

D_6 **IF** *PCTS_mlock* or *PCTS_munlock* and a PCD.1b documents the following **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: A PCD.1b that documents the implementation requires memory locking only in multiples of {PAGESIZE}, does so in 12.1.2.4.

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_OPTION

17 **IF** *PCTS_ mlock* and *PCTS_MULTIPLE_OF_PAGESIZE* and *PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE* **THEN**

IF *PCTS_GAP_mlock* **THEN**

TEST: A call to *mlock*(), when the *addr* argument is not a multiple of the page size {PAGESIZE}, returns a value of -1 and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_MULTIPLE_OF_PAGESIZE* and *PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE* and *PCTS_ munlock* **THEN**

TEST: A call to *munlock*(), when the *addr* argument is not a multiple of the page size {PAGESIZE}, returns a value of -1 and sets *errno* to [EINVAL].

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_OPTION

D_7 **IF** *PCTS_mlock* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the implementation-defined limit on an amount of memory that a process may lock, does so in 12.1.2.4.

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_OPTION

19 **IF** *PCTS_ mlock* and *PCTS_DETECT_LOCKABLE_MEMORY_LIMIT_ mlock* **THEN**

IF *PCTS_GAP_mlock* **THEN**

TEST: A call to *mlock*(), when locking the pages mapped by the specified range would exceed an implementation-defined limit on the amount of memory that the process may lock, returns a value of -1 and sets *errno* to [ENOMEM].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

D_8 **IF** *PCTS_mlock* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the implementation detects whether or not a process has appropriate privileges to lock pages, does so in 12.1.2.4.

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_OPTION

20 **IF** *PCTS_ mlock* and *PCTS_DETECT_NO_AP* **THEN**

IF *PCTS_RAP_mlock* **THEN**

TEST: A call to *mlock*(), when the calling process does not have the appropriate privilege to perform the requested operation , returns a value of -1 and sets *errno* to [EPERM].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

12.2 Memory Mapping Functions

12.2.1 Map Process Address to a Memory Object

Function: *mmap* ().

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

1

M_GA_stdC_proto_decl(void *; mmap; void*addr, size_tlen; int prot, int flags, int fildes, off_t off; sys/mman.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for mlock: PASS[1, 2], NO_OPTION

2

M_GA_commonC_result_decl (void*; mmap; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for mlock: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; mlock; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sem_init: PASS, NO_OPTION

4

M_GA_macro_args (mmap; sys/mman.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mlock: PASS, NO_OPTION

mmap **IF** *PCTS_mmap* **THEN**
TEST: A successful call to the function *mmap*() establishes a mapping between the address space of the process for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off* for *len* bytes, and returns the address at which the mapping was placed.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

D_1 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the implementation-dependent function of the parameter *addr* and the values of *flags*, which determines the address at which the mapping is placed, does so in 12.1.2.2.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

5 **IF** *PCTS_mmap* **THEN**
TEST: The address range starting at the address returned by *mmap(addr, len, prot, flags, fildes, off)*, and continuing for *len* bytes, is legitimate for the possible (not necessarily current) address space of the process.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

6 **IF** *PCTS_mmap* **THEN**
TEST: In a call to *mmap(addr, len, prot, flags, fildes, off),* the range of bytes starting at *off* and continuing for *len* bytes is legitimate for the possible (not necessarily current) offsets in the file or, for the shared memory object represented by *fildes*.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

7 **IF** *PCTS_mmap* **THEN**
TEST: The mapping established by *mmap(addr, len, prot, flags, fildes, off)* replaces any previous mappings for those whole pages containing any part of the address space of the process, starting at the address returned by *mmap*() and continuing for *len* bytes.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8 **IF** *PCTS_mmap* **THEN**
TEST: In a call to *mmap(addr, len, prot, flags, fildes, off)*, the parameter *prot* determines whether read, write, execute, or some combination of accesses are permitted to the data being mapped.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

prot_values

IF *PCTS_mmap* **THEN**
SETUP: Include the header `<sys/mman.h>`.
TEST: The constants PROT_NONE, PROT_READ, PROT_WRITE, and PROT_EXEC are defined and are bitwise distinct.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

9 **IF** *PCTS_mmap* **THEN**
TEST: In the call *mmap(addr, len, prot, flags, fildes, off)*, *prot* may be either PROT_NONE, or the bitwise inclusive OR of one or more of the flags PROT_READ, PROT_WRITE, PROT_EXEC, and PROT_NONE.
TR: Try PROT_NONE and all eight bitwise combinations of the other three values.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

R_1 **IF** *PCTS_mmap* **THEN**
TEST: When the combination of access types specified by *prot* is not supported, the call *mmap(addr, len, prot, flags, fildes, off)* fails.
ELSE *NO_OPTION*
SEE: Assertion mmap_ENOTSUP in 12.1.2.2.

10 **IF** *PCTS_ mmap* **THEN**
IF {POSIX_MEMORY_PROTECTION} **THEN**
TEST: When PROT_WRITE is unset, writes to the region mapped by *mmap*() fail.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

mem_protect_flags

11 **IF** *PCTS_ mmap* **THEN**
IF {POSIX_MEMORY_PROTECTION} **THEN**
TEST: When PROT_NONE alone has been set, writes to the region mapped by *mmap(addr, len, prot, flags, fildes, off)* fail.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

mem_protect_flags
IF *PCTS_ mmap* **THEN**
IF {POSIX_MEMORY_PROTECTION} **THEN**
TEST: The implementation supports the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the inclusive OR of PROT_READ and PROT_WRITE
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

12 **IF** *PCTS_ mmap* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: After a call to *mmap(addr, len, prot, flags, fildes, off),* with the flag MAP_SHARED SET, write references change the underlying object.

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_ mmap* **THEN**

IF *PCTS_MAP_PRIVATE* **THEN**

TEST: After a call to *mmap(addr, len, prot, flags, fildes, off)*, with the flag MAP_PRIVATE set, modifications to the mapped data by the calling process are visible only to the calling process and do not change the underlying object.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether modifications to the underlying object, done after the MAP_PRIVATE mapping is established, are visible through the MAP_PRIVATE mapping, does so in 12.1.2.2.

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

14 **IF** *PCTS_ mmap* **THEN**

IF *PCTS_MAP_PRIVATE* **THEN**

TEST: MAP_SHARED must be specified in a call to the *mmap*() function.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_ mmap* **THEN**

TEST: When *mmap()* is called, the mapping type is retained across *fork*().

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

16 **IF** *PCTS_ mmap* **THEN**

IF *PCTS_MAP_FIXED* **THEN**

TEST: When MAP_FIXED is set, the address returned by *mmap(addr, len, prot, flags, fildes, off)* is *addr* exactly.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

D_3 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether MAP_FIXED is supported, does so in 12.1.2.2.

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

D_4 **IF** *PCTS_mmap* **THEN**

TEST: A PCD.1b documents how the system uses *addr* to arrive at *pa*, when MAP_FIXED is not set, in 12.1.2.2.

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

17 **IF** *PCTS_ mmap* **THEN**

TEST: When MAP_FIXED is not set, a call to *mmap()* never places a mapping at address 0.

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

18 **IF** *PCTS_mmap* **THEN**
TEST: When MAP_FIXED is not set, a call to *mmap()* never replaces an extant mapping.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

19 **IF** *PCTS_mmap* **THEN**
IF *PCTS_MAP_FIXED* **THEN**
TEST: When MAP_FIXED is specified and *addr* is non0, the address returned by *mmap(addr, len, prot, flags, fildes, off)* has the same remainder as the *off* parameter, modulo the page size {PAGESIZE}.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

20 **IF** *PCTS_mmap* **THEN**
IF *PCTS_MULTIPLE_OF_PAGESIZE* **THEN**
TEST: The argument *off* must be a multiple of the page size.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

D_5 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents that the argument *off* , in a call to *mmap(addr, len, prot, flags, fildes, off),* must be a multiple of the page size, does so in 12.1.2.2.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

21 **IF** *PCTS_mmap* **THEN**
IF *PCTS_MULTIPLE_OF_PAGESIZE* and *PCTS_MAP_FIXED* **THEN**
TEST: When MAP_FIXED is specified, the argument *addr* must be a multiple of the page size.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

D_6 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents that the argument *addr* , in a call to *mmap(addr, len, prot, flags, fildes, off)* with MAP_FIXED specified, must be a multiple of the page size, does so in 12.1.2.2.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

22 **IF** *PCTS_mmap* **THEN**
IF *PCTS_MAP_FIXED* **THEN**
TEST: When MAP_FIXED is specified, the parameter *len* need not meet a size or alignment constraint.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

23 **IF** *PCTS_mmap* **THEN**
IF *PCTS_MAP_FIXED* **THEN**
TEST: When MAP_FIXED is specified, after a call to *mmap(addr, len, prot, flags, fildes, off),* any partial page specified by the address range starting at the returned address and continuing for *len* bytes, is included in the object.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

24 **IF** *PCTS_mmap* **THEN**
TEST: Any partial page at the end of an object is zero-filled.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

25 **IF** *PCTS_mmap* **THEN**
TEST: Modified portions of the last page of an object that are beyond its end are not written out.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

mmap_SIGBUS
IF *PCTS_mmap* **THEN**
IF {POSIX_MEMORY_PROTECTION} **THEN**
TEST: Following a call to *mmap(addr, len, prot, flags, fildes, off)*, references within the address range starting at the returned address and continuing for *len* bytes to whole pages following the end of an object, result in delivery of a SIGBUS signal.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

D_7 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of references within the address range starting at *pa* and continuing for *len* bytes, to whole pages following the end of an object, when the {_POSIX_MEMORY_PROTECTION} option is not supported, does so in 12.1.2.2.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

D_8 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mmap*() function, does so in 12.1.2.2.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

12.2.1.3 Returns

R_2 **IF** *PCTS_mmap* **THEN**
TEST: When a call to *mmap( )* completes successfully, the interface returns the address at which the mapping was placed.
ELSE *NO_OPTION*
SEE: Assertion mmap in 12.2.1.2.

R_3 **IF** *PCTS_mmap* **THEN**
TEST: When a call to *mmap( )* completes unsuccessfully, the interface returns a value of MAP_FAILED and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: Assertion mmap in 12.2.1.4.

26 **IF** *PCTS_mmap* **THEN**
SETUP: Include the header `<sys/mman.h>`.
TEST: The symbol MAP_FAILED is defined.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

R_4 **IF** *PCTS_mmap* **THEN**
TEST: No successful return from *mmap*() returns the value of MAP_FAILED.
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SEE: Assertion mmap in 12.2.1.4.

12.2.1.4 Errors

27 **IF** *PCTS_mmap* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when the file descriptor *fildes* is not open for read, returns a value of MAP_FAILED and sets *errno* to [EACCESS].
TR: Specify each possible protection.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

28 **IF** *PCTS_mmap* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when the file descriptor *fildes* is not open for write and PROT_WRITE is specified for a MAP_SHARED type mapping, returns a value of MAP_FAILED and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

29 **IF** *PCTS_mmap* **THEN**
TEST: A call to *mmap()*, when the mapping could not be locked in memory, if required by *mlockall*(), due to a lack of resources, returns a value of MAP_FAILED and sets *errno* to [EAGAIN].
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

30 **IF** *PCTS_mmap* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when the *fildes* argument is not a valid open file descriptor, returns a value of MAP_FAILED and sets *errno* to EBADF].
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

31 **IF** *PCTS_mmap* **THEN**
IF *PCTS_DETECT_INVALID_FLAGS_mmap* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when the value in *flags* is invalid (e.g., neither MAP_PRIVATE or MAP_SHARED is set), returns the value of MAP_FAILED and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

32 **IF** *PCTS_mmap* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when the *fildes* argument refers to an object for which *mmap*() is meaningless, such as a terminal, returns a value of MAP_FAILED and sets *errno* to [ENODEV].
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

33 **IF** *PCTS_mmap* **THEN**
IF *PCTS_MAP_FIXED* **THEN**
TEST: *A call to* mmap(addr, len, prot, flags, fildes, off), *when MAP_FIXED is* specified, and the address range starting at *addr* and continuing for *len* bytes exceeds that allowed for the address space of a process, or MAP_FIXED is not specified, and there is insufficient room in the address space to effect the mapping; returns a value of MAP_FAILED and sets *errno* to [ENOMEM].
There is no known reliable test method for this assertion.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

34 **IF** *PCTS_mmap* **THEN**
IF *PCTS_mlockall PCTS_GAP_mlockall* **THEN**
TEST: A call to *mmap()*, when the mapping could not be locked in memory, if required by *mlockall*(), because it would require more space than the system is able to supply, returns a value of MAP_FAILED and sets *errno* to [ENOMEM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

35 **IF** not *PCTS_mmap* **THEN**
TEST: A call to mmap(), returns a value of MAP_FAILED and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

36 **IF** *PCTS_mmap* **THEN**
IF not *PCTS_MAP_FIXED* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when MAP_FIXED is specified in the *flags* argument, returns a value of MAP_FAILED and sets *errno* to [ENOTSUP].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

37 **IF** *PCTS_mmap* **THEN**
IF not *PCTS_MAP_PRIVATE* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when MAP_PRIVATE is specified in the *flags* argument, returns a value of MAP_FAILED and sets *errno* to [ENOTSUP].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

mmap_ENOTSUP
IF *PCTS_mmap* **THEN**
TEST: A call to *mmap()*, when the implementation does not support the combination of accesses requested in the *prot* argument, returns a value of MAP_FAILED and sets *errno* to [ENOTSUP].
TR: Test with MAP_SHARED and MAP_PRIVATE both in the *prot* argument.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

38 **IF** *PCTS_mmap* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when the addresses in the range starting at *off* and continuing for *len* bytes are invalid for the object specified by *fildes*, returns a value of MAP_FAILED and sets *errno* to [ENXIO].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

39 **IF** *PCTS_mmap* **THEN**
IF *PCTS_MAP_FIXED* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when MAP_FIXED is specified in *flags* and the combination of *addr*, *len,* and *off* is invalid for the object specified by *fildes*, returns a value of MAP_FAILED and sets *errno* to [ENXIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST SUPPORT, NO_OPTION

40 **IF** *PCTS_ mmap* **THEN**

IF *PCTS_MULTIPLE_OF_PAGESIZE* **THEN**

TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when the argument *off* is not a multiple of the page size {PAGESIZE}, returns a value of MAP_FAILED and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_ SUPPORT, NO_OPTION

41 **IF** *PCTS_ mmap* **THEN**

IF *PCTS_MULTIPLE_OF_PAGESIZE* and *PCTS_MAP_FIXED* **THEN**

TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when MAP_FIXED is specified and the argument *addr* is not a multiple of the page size {PAGESIZE}, returns a value of MAP_FAILED and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_ SUPPORT, NO_OPTION

12.2.2.2 Unmap Previously Mapped Addresses

function: *munmap*().

12.2.2.1 Synopsis

1

M_GA_stdC_proto_decl(int;; munmap; void*addr, size_t len; sys/mman.h;;;)

SEE: Assertion GA_stdC_proto_decl in 2.7.3.

Conformance for munmap: PASS[1, 2], NO_OPTION

2

M_GA_commonC_result_decl (munmap; sys/mman.h;;;)

SEE: Assertion GA_commonC_int_result_decl in 2.7.3.

Conformance for munmap: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; munmap; sys/mman.h;;;)

SEE: Assertion GA_macro_result_decl in 1.3.4.

Conformance for munmap: PASS, NO_OPTION

4

M_GA_macro_args (munmap; sys/mman.h;;;)

SEE: Assertion GA_macro_args in 2.7.3.

Conformance for munmap: PASS, NO_OPTION

12.2.2.2 Description

munmap

IF *PCTS_ munmap* **THEN**

TEST: A successful call to the function *munmap(addr, len)* removes any mappings for those entire pages containing any part of the address space of the process, starting at *addr* and continuing for *len* bytes, and returns the value 0.

ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for munmap: PASS, NO_OPTION

munmap_SIGSEV

IF *PCTS_ munmap* **THEN**

TEST: Following a successful call to *munmap()*, references to the unmapped pages result in the delivery of a SIGSEV signal to the process.

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_OPTION

5 **IF** *PCTS_ munmap* **THEN**

TEST: When there are no mappings in the specified address range, then *munmap()* has no effect.

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_OPTION

6 **IF** *PCTS_ munmap* **THEN**

IF *PCTS_MULTIPLE_OF_PAGESIZE* **THEN**

TEST: In a call to *munmap(addr, len)*, the argument *addr* must be a multiple of the page size {PAGESIZE}.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_munmap* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents that the argument *addr,* in a call to *munmap*(*addr, len*), must be a multiple of the page size {PAGESIZE}, does so in 12.2.2.2.

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_OPTION

7 **IF** *PCTS_ munmap* **THEN**

IF *PCTS_munmap* and *PCTS_MAP_PRIVATE* **THEN**

TEST: When a mapping to be removed is private, any modifications made in this address range are discarded.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_TEST_SUPPORT, NO_OPTION

munlock_remove_maps

IF *PCTS_ munmap* **THEN**

IF *PCTS_mlock* and *PCTS_GAP_MLOCK* and *PCTS_mlockall* and *PCTS_GAP_MLOCKALL and PCTS_munlock* **THEN**

TEST: Following a call to *munmap(addr, len,)*, any memory locks (see POSIX.1b {3} 12.1 2 and POSIX.1b {3} 12.1 1) associated with this address range are removed, as if by an appropriate call to *munlock*().

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_TEST_ SUPPORT, NO_OPTION

D_2 **IF** *PCTS_munmap* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the behavior of this function if the mapping was not established by a call to *munmap*(), does so in 12.2.2.2.

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_OPTION

D_3 **IF** *PCTS_munmap* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *munmap*() function does so in 12.2.2.2.

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12.2.2.3 Returns

R_1 IF *PCTS_munmap* **THEN**
TEST: When a call to *munmap*() completes successfully, the interface returns to a value of 0.
ELSE *NO_OPTION*
SEE: Assertion munmap in 12.2.2.2.

R_2 IF *PCTS_munmap* **THEN**
TEST: When a call to *munmap*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 12.2.2.4.

12.2.2.4 Errors

8 **IF** *PCTS_ munmap* **THEN**
TEST: A call to *munmap*(), when some of the addresses in the range starting at *addr* and continuing for *len* bytes are outside the range allowed for the address space of a process, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for munmap: PASS, NO_TEST, NO_OPTION

9 **IF** not *PCTS_ munmap* **THEN**
TEST: A call to *munmap*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for munmap: PASS, NO_OPTION

10 **IF** *PCTS_ munmap* **THEN**
IF *PCTS_MULTIPLE_OF_PAGESIZE*
and *PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE* **THEN**
TEST: A call to *munmap(addr, len,)*, when the value of *addr* is not a multiple of the page size {PAGESIZE}, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for munmap: PASS, NO_TEST_ SUPPORT, NO_OPTION

12.2.3 Change Memory Protection

Function: *mprotect*().

12.2.3.1 Synopsis

1
M_GA_stdC_proto_decl(int; mprotect; const void*addr, size_t len; int_prot; sys/mman.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for mprotect: PASS[1, 2], NO_OPTION

2
M_GA_commonC_result_decl (mprotect; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for munmap: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; mprotect; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for mprotect: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4 ***M_GA_macro_args (mprotect; sys/mman.h;;;)***
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mprotect: PASS, NO_OPTION

12.2.3.2 Description

mprotect

IF *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A successful call to the function *mprotect(addr, len, prot)* changes the access protections to those specified by *prot* for whole pages containing any part of the address space of the process, starting at address *addr* and continuing for *len* bytes, and returns the value 0.
TR: Test for PROT_READ, PROT_WRITE, and PROT_NONE individually.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

R-1 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*(). Include the header `<sys/mman.h>`.
TEST: The constants PROT_NONE,PROT_READ, PROT_WRITE, and PROT_EXEC are defined and are bitwise distinct.
ELSE *NO_OPTION*
SEE: Assertion prot_values in 12.2.1.2
Conformance for mprotect: PASS, NO_OPTION

5 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: In the call *mprotect(addr, len, prot)*, the only permitted values for *prot* are PROT_NONE or the bitwise inclusive OR of one or more of the values PROT_READ, PROT_WRITE..
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

R-2 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When an implementation cannot support the combination of access types specified by *prot*, the call to *mprotect*() fails.
ELSE *NO_OPTION*
SEE: Assertion mprotect_ENOTSUP in 12.2.3.4.

D_1 **IF** *PCTS_mprotect* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether accesses other than those specified by *prot* are permitted, does so in 12.2.3.2.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

6 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: All accesses fail where PROT_WRITE alone has been set.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

7 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: All accesses fail where PROT_NONE alone has been set.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: The implementation supports at least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the inclusive OR of PROT_READ and PROT_WRITE.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

9 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When MAP_PRIVATE is not specified in the original mapping, and PROT_WRITE is specified, mapped objects are opened in the specified address range with write permission.
TR: Try with and without closing the file descriptors used to map the objects after performing the original mapping.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

D_2 **IF** *PCTS_mprotect* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents that the argument *addr* in a call to *mprotect(addr, len, prot)* must be a multiple of the page size {PAGESIZE}, does so in 12.2.3.2.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

D_3 **IF** *PCTS_mprotect* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of *mprotect(),* if the mapping was not established by a call to *mmap*(), does so in 12.2.3.2.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

D_4 **IF** *PCTS_mprotect* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mprotect*() function does so in 12.2.3.2.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

12.2.3.3 Returns

R_3 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When a call to *mprotect*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion mprotect in 12.2.3.2.

R_4 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When a call to *mprotect*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 12.2.3.4.

10 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When *mprotect*() fails and returns [EINVAL], the protections in the pages in the address range starting at *addr* and continuing for *len* bytes are unchanged.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12.2.3.4 Errors

11 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect*(), when the memory object was not opened for read, regardless of the protection specified, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

12 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect(addr, len, prot)*, when the memory object was not opened for write, and PROT_WRITE is specified for a MAP_SHARED type mapping, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

13 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
IF *PCTS_MAP_PRIVATE* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect(addr, len, prot)*, when the *prot* argument specifies PROT_WRITE on a MAP_PRIVATE mapping, and there are insufficient memory resources to reserve for locking the private pages, if required, returns a value of -1 and sets *errno* to [EAGAIN].
NOTE: There is no known reliable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_TEST, NO_TEST_ SUPPORT, NO_OPTION

14 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect(addr, len, prot)*, when the addresses in the range starting at *addr* and continuing for *len* bytes are outside the range allowed for the address space of a process, returns a value of -1 and sets *errno* to [ENOMEM].
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

15 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect(addr, len, prot)*, when the addresses in the range starting at *addr* and continuing for *len* bytes specify one or more pages that are not mapped, returns a value of -1 and sets *errno* to [ENOMEM].
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

16 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
IF *PCTS_MAP_PRIVATE* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect(addr, len, prot)*, when the *prot* argument specifies PROT_WRITE on a MAP_PRIVATE mapping, and it would require more space than the system is able to supply for locking the private pages, if required, returns a value of -1 and sets *errno* to [ENOMEM].

ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_TEST, NO_TEST_ SUPPORT, NO_OPTION

17 **IF** not *PCTS_mprotect* **THEN**
TEST: A call to *mprotect()* returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

mprotect_ENOTSUP

IF *PCTS_mprotect* and *PCTS_mmap* **THEN**

SETUP: Map memory pages using the function *mmap*().

TEST: A call to *mprotect()*, when the implementation does not support the combination of accesses requested in the *prot* argument, returns a value of -1 and sets *errno* to [ENOTSUP].

ELSE *NO_OPTION*

Conformance for mprotect: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**

IF *PCTS_MULTIPLE_OF_PAGESIZE*
and *PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE* **THEN**

SETUP: Map memory pages using the function *mmap*().

TEST: A call to *mprotect(addr, len, prot)*, when the value of *addr* is not a multiple of the page size {PAGESIZE}, returns a value of -1 and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mprotect: PASS, NO_TEST_ SUPPORT, NO_OPTION

12.2.4 Memory Object Synchronization

Function: *msync*().

12.2.4.1 Synopsis

1

M_GA_stdC_proto_decl(int; msync; void*addr, size_t len; int flags; sys/mman.h;;;)

SEE: Assertion GA_stdC_proto_decl in 2.7.3.

Conformance for msync: PASS[1, 2], NO_OPTION

2

M_GA_commonC_result_decl (msync; sys/mman.h;;;)

SEE: Assertion GA_commonC_int_result_decl in 2.7.3.

Conformance for msync: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; msync; sys/mman.h;;;)

SEE: Assertion GA_macro_result_decl in 1.3.4.

Conformance for msync: PASS, NO_OPTION

4

M_GA_macro_args (msync; sys/mman.h;;;)

SEE: Assertion GA_macro_args in 2.7.3.

Conformance for msync: PASS, NO_OPTION

12.2.4.2 Description

msync **IF** *PCTS_msync* and *PCTS_munmap* **THEN**

IF: *PCTS_msync_storage* **THEN**

SETUP: Map memory pages using the function *mmap*()

TEST: A successful call to the function *msync*() writes all modified data to permanent storage locations in those whole pages containing any part of the address space of the process, starting at address *addr* and continuing for *len* bytes, and returns the value 0.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5 **IF** *PCTS_msync* and *PCTS_munmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A successful call to the function *msync*(*addr, len, flags*), where the *flags* argument contains the value MS_INVALIDATE, invalidates cached copies of the data.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST, NO_OPTION

D_1 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents a call to *msync*(*addr, len, flags*), when there are no permanent storage locations to write the modified data, does so in 12.2.4.2.
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

D_2 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents that the argument *addr,* in a call to *msync*(*addr, len, flags*), must be a multiple of the page size {PAGESIZE}, does so in 12.2.4.2.
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

D_3 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the implementation also writes out other file attributes, does so in 12.2.4.2.
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

6 **IF** *PCTS_msync* and *PCTS_munmap* **THEN**
IF *PCTS_msync_storage* and *PCTS_MAP_PRIVATE* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: When the *msync*() function is called on MAP_PRIVATE mapping, any modified data is not written to the underlying object, nor is such data made visible to other processes.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST, NO_OPTION

D_4 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether data in MAP_PRIVATE mappings has any permanent storage locations, does so in 12.2.4.2.
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

D_5 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect of *msync*() on shared memory objects, does so in 12.2.4.2.
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

7 **IF** *PCTS_msync* **THEN**
SETUP: Include the header `<sys/msync.h>`.
TEST: The constants MS_ASYNC, MS_SYNC, and MS_INVALIDATE are defined and are bitwise distinct.
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

8 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When MS_ASYNC is speciufied, *msync*() returns immediately once all the write operations are initiated or queued for servicing.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When MS_SYNC is specified, *msync*() does not return until all write operations are completed as defined for synchronized I/O data integrity completion.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWrite in 2.2.119.
Conformance for msync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

10 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When *msync()* is called, with MS_ASYNC specified, all write operations are completed as defined for synchronized I/O data integrity completion.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWrite in 2.2.119.
Conformance for msync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

R_1 **IF** *PCTS_msync* **THEN**
TEST: A call to *msync*() cannot specify both MS_ASYNC and MS_SYNC.
ELSE *NO_OPTION*
SEE: Assertions **msync_einval** in 12.2.4.4 .

11 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: Following a call to *msync*(*addr, len, flags*) with MS_INVALIDATE set, references to the object obtain data that was consistent with the permanent storage locations sometime between the call to *msync*() and the first subsequent memory reference to the data.
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

D_6 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of this function, if the mapping was not established by a call to *mmap*(), does so in 12.2.4.2.
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

D_7 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *msync*() function, does so in 12.2.4.2.
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for msync: PASS, NO_OPTION

12.2.4.3 Returns

R_2 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: When a call to *msync*() completes successfully, the interface returns a value of 0.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion msync in 12.2.4.2.

R_3 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: When a call to *msync*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion msync in 12.2.4.4

12.2.4.4 Errors

12 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *msync*(*addr, len, flags*), when some or all of the addresses in the range starting at *addr* and continuing for *len* bytes are locked, and MS_INVALIDATE is specified, returns a value of -1 and sets *errno* to [EBUSY].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

msync_einval
IF *PCTS_msync* and *PCTS_mmap* **THEN**
IF: *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *msync*(*addr, len, flags*), when the value in *flags* is invalid, returns a value of -1 and sets *errno* to [EINVAL].
TR: Test with *flags* having both MS_ASYNC and MS_SYNC.
NOTE: A subroutine is recommended that either returns a *flags* argument that includes unimplemented flags, or indicates that there is no way to generate a *flags* argument that includes unimplemented flags on the system.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *msync*(*addr, len, flags*), when the addresses in the range starting at *add* and continuing for *len* bytes are outside the range allowed for the address space of a process, returns a value of -1 and sets *errno* to [ENOMEM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

14 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *msync*(*addr, len, flags*), when the addresses in the range starting at *addr* and continuing for *len* specify one of more pages that are not mapped, returns a value of -1 and sets *errno* to [ENOMEM].
TR: Test for one page not mapped and for more than one page not mapped.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** not *PCTS_msync* **THEN**
TEST: A call to *msync*(), returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

16 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* and *PCTS_MULTIPLE_OF_PAGESIZE* and *PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *msync*(*addr, len, flags*), when the value of *addr* is not a multiple of the page size {PAGESIZE}, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

12.3 Shared Memory Functions

12.3.1 Open a Shared Memory Object

Function: *shm_open*().

12.3.1.1 Synopsis

1
M_GA_stdC_proto_decl(int; shm_open; const char*name, int oflag, mod_t mode; sys/mman.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for shm_open: PASS[1, 2], NO_OPTION

2
M_GA_commonC_result_decl (shm_open; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for shm_open:: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; shm_open; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for shm_open: PASS, NO_OPTION

4
M_GA_macro_args (shm_open; sys/mman.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for shm_open: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12.3.1.2 Description

shm_open

IF *PCTS_shm_open* **THEN**
TEST: A successful call to the *shm_open*() function creates an open file description that refers to the shared memory object, and returns a valid file descriptor that refers to that open file description.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

5 **IF** *PCTS_shm_open* **THEN**
TEST: The file descriptor returned by *shm_open*() is a nonnegative integer.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_1 **IF** *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the shared memory object *name* appears in the file system, and is visible to other functions that take pathnames as arguments, does so in 12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

6 ***M_GA_portableFilenames(shm_open)***
SEE: Assertion GA_portableFilenames in 2.2.4.0.
Conformance for shm_open: PASS, NO_OPTION

7 ***M_GA_upperLowerNames (shm_open)***
SEE: Assertion GA_upperLowerNames in 2.2.2.40.
Conformance for shm_open:: PASS, NO_OPTION

8 ***M_GA_PRNoTrunc(shm_open)***
SEE: Assertion GA_PRNoTrunc in 2.3.6.
Conformance for shm_open: PASS, NO_OPTION

9 ***M_GA_PRNoTruncError (shm_open)***
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for shm_open: PASS, NO_OPTION

10 **IF** *PCTS_shm_open* **THEN**
TEST: When *name* begins with the slash character, then processes calling *shm_open*(*name, oflag, mode*) with the same value of *name* refer to the same shared memory object, as long as that name has not been removed.
TR: Test using two different processes.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_2 **IF** *PCTS_shm_open* **THEN**
TEST: A PCD.1b that documents the effect if the *name* to *shm_open(name, oflag, mode)* does not begin with the slash character, does so in 12.3.1.2.is specified
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_3 **IF** *PCTS_shm_open* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: A PCD.1b that documents the interpretation of slash characters other than the leading slash character in the *name* argument to *shm_open(name, oflag, mode)*, does so in 12.3.1.2.

ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

11 **IF** *PCTS_shm_open* **THEN**
TEST: A successful call to *shm_open*() returns a file descriptor for the shared memory object that is the lowest-numbered file descriptor not currently open for that process.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

12 **IF** *PCTS_shm_open* **THEN**
TEST: The process does not share the open file description created by *shm_open*() with any other processes.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_4 **IF** *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the file offset is set by *shm-open*(), does so in 12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

13 **IF** *PCTS_shm_open* **THEN**
TEST: The FD_CLOEXEC file descriptor flag associated with the new file descriptor is set by *shm_open*().
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

14 **IF** *PCTS_shm_open* **THEN**
TEST: Following a call to *shm_open(name, oflag, mode)*, the file status flags and file access modes of the open file description are set according to the value of *oflag*.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

15 **IF** *PCTS_shm_open* **THEN**
SETUP: Include the header `<sys/mman.h>`.
TEST: The constants O_RDONLY, O_RDWR, O_CREAT, O_EXCL and O_TRUNC are defined and have the same values as defined in the header `<fcntl.h>`.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

16 **IF** *PCTS_shm_open* **THEN**
SETUP: Include the header `<sys/mman.h>`.
TEST In a call to *shm_open(name, oflag, mode),* the *oflag* argument must be the bitwise inclusive OR of one or more of the following flags: O_RDONLY, O_RDWR, O_CREAT, O_EXCL, or O_TRUNC.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

17 **IF** *PCTS_shm_open* **THEN**
TEST: In the call to *shm_open(name, oflag, mode), oflag* must have either O_RDONLY or O_RDWR set.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18 **IF** *PCTS_shm_open* **THEN**
TEST: In the call to *shm_open(name, oflag, mode), oflag* must have both O_RDONLY, or O_RDWR set.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

19 **IF** *PCTS_shm_open* **THEN**
TEST: In the call to *shm_open(name, oflag, mode),* any combination of the following flags: O_CREAT, O_EXCL, and O_TRUNC, may be specified in *oflag*.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

20 **IF** *PCTS_shm_open* **THEN**
TEST: In a call to *shm_open(name, oflag, mode)*, the flag O_CREAT has no effect when it refers to a shared memory object that was previously created with the O_EXCL flag unset.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

21 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open(name, oflag, mode)* creates a shared memory object when the *flag* O_CREAT is set, and the shared memory object did not exist previously.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

22 **IF** *PCTS_shm_open* **THEN**
SETUP: Create a shared memory object by calling *shm_open (name, oflag, mode)* with the *oflag* argument set to O_CREAT.
TEST: The user ID of a newly created, shared memory object is set to the effective user ID of the process.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

23 **IF** *PCTS_shm_open* **THEN**
SETUP: Create a shared memory object by calling *shm_open (name, oflag, mode)* with the *oflag* argument set to O_CREAT.
TEST: The group ID of a newly created, shared memory object is set to the effective user ID of the process.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

24 **IF** *PCTS_shm_open* **THEN**
TEST: When a shared memory object is created by calling *shm_open(name, oflag, mode)*, with the *oflag* argument set to O_CREAT, the permission bits are set to the value of the *mode* argument, except for those set in the file mode creation mask of the process.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_5 **IF** *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect when bits in *mode*, other than the file permission bits, are set during creation of a shared memory object by a call to *shm-open*(*name, oflag, mode*)does so in 12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

25 **IF** *PCTS_shm_open* **THEN**
TEST: In the call *shm_open(name, oflag, mode)*, the *mode* argument does not affect whether the shared memory object is opened for reading, or writing, or both.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TR: Try all three, with all possibly mode bit combinations.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

26 **IF** *PCTS_shm_open* **THEN**
TEST: A newly created shared memory object has a size of 0
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

R_1 **IF** *PCTS_shm_open* **THEN**
TEST: When O_EXCL and O_CREAT are set, and the shared memory object already exists, *shm_open*() fails.
ELSE *NO_OPTION*
SEE: Assertion shm_exist_err in 12.3.1.4.

27 **IF** *PCTS_shm_open* **THEN**
TEST: The check by *shm_open()* for the existence of the shared memory object, and the creation of the object if it does not exist, are atomic with respect to other processes executing *shm_open*() with O_EXCL and O_CREAT set, and naming the same shared memory object.
NOTE: There is no known reliable test method for this assertion.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_TEST, NO_OPTION

D_6 **IF** *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of a call to *shm-open*(*name, oflag, mode*), if O_EXCL is set and O_CREAT is not set, does so in 12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

28 **IF** *PCTS_shm_open* **THEN**
TEST: When a shared memory object exists, a successful call to *shm_open(name, oflag, mode)*, specifying both O_RDWR and O_TRUNC, truncates the object to zero length.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

29 **IF** *PCTS_shm_open* **THEN**
TEST: When a shared memory object exists, a successful call to *shm_open(name, oflag, mode),* specifying both O_RDWR and O_TRUNC, leaves the mode and owner unchanged.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_7 **IF** *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of specifying both O_TRUNC and O_RDONLY when calling *shm-open*(*name, oflag, mode*), does so in 12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

30 **IF** *PCTS_shm_open* **THEN**
TEST: When a shared memory object is created by *shm_open( )*, the state of the shared memory object, including all data associated with the shared memory object, persists until the shared memory object is unlinked and all other references are gone.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D_8 IF *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the name and state of a shared memory object remain valid after a system reboot does so in 12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_9 IF *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *shm-open*() function, does so in 12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

12.3.1.3 Returns

R_2 IF *PCTS_shm_open* **THEN**
TEST: When a call to *shm_open*() completes successfully, it returns a nonnegative integer representing the lowest numbered unused file descriptor.
ELSE *NO_OPTION*
SEE: Assertion shm_open in 12.3.1.2.

R_3 IF *PCTS_shm_open* **THEN**
TEST: When a call to *shm_open*() completes unsuccessfully, it returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 12.3.1.4.

12.3.1.4 Errors

31 IF *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when the shared memory object exists and the permission specified by *oflag* is denied, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

32 IF *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when the shared memory object does not exist and permission to create the shared memory object is denied, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

33 IF *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when O_TRUNC is specified and write permission is denied, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

shm_exist_err
IF *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when O_CREAT and O_EXCL are set and the named shared memory object already exists, returns a value of -1 and sets *errno* to [EEXIST].
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

34 IF *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when the *shm_open*() operation is interrupted by a signal, returns a value of -1 and sets *errno* to [EINTR].

Copyright © 2003 IEEE – All rights reserved
--`,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

35 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when the *shm_open*() operation is not supported for the given name, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns a name for which *shm_open* is not supported, or indicates that there is no way to generate a name for which *shm_open*() is not supported on the system.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_10 **IF** *PCTS_shm_open* **THEN**
TEST: A PCD.1b that documents under what circumstances [EINVALll] may be returned, does so in 12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

36 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when too many file descriptors are currently in use by this process, returns a value of -1 and sets *errno* to [EMFILE].
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

37 **IF** *PCTS_shm_open* **THEN**
IF {PATH_MAX} <= *PCTS_PATH_MAX* **THEN**
TEST: A call to *shm_open( )*, when the length of the *name* string exceeds {PATH_MAX}, returns a value of -1 and sets *errno* to [ENAMETOOLONG].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_TEST_SUPPORT, NO_OPTION

38 **IF** *PCTS_shm_open* and {_POSIX_NO_TRUNC} **THEN**
IF {NAME_MAX} <= *PCTS_NAME_MAX* **THEN**
TEST: A call to *shm_open( )*, when a pathname component is longer than {NAME_MAX}, returns a value of -1 and sets *errno* to [ENAMETOOLONG].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_TEST_SUPPORT, NO_OPTION

39 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when too many shared memory objects are currently open in the system, returns a value of -1 and sets *errno* to [EINFILE].
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

40 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when O_CREAT is not set and the named shared memory object does not exist, returns a value of -1 and sets *errno* to [ENOENT].
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

41 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when there is insufficient space for the creation of the new shared memory object, returns a value of -1 and sets *errno* to [ENOSPC].
NOTE: There is no known reliable test method for this assertion.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

42 **IF** not *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

12.3.2 Remove a Shared Memory Object

Function: *shm_unlink*().

12.3.2.1 Synopsis

1
M_GA_stdC_proto_decl(int; shm_unlink; const char*name,;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for shm_unlink: PASS[1, 2], NO_OPTION

2
M_GA_commonC_result_decl (shm_unlink;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for shm_unlink:: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; shm_unlink;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for shm_open: PASS, NO_OPTION

4
M_GA_macro_args (shm_unlink;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for shm_open: PASS, NO_OPTION

12.3.2.2 Description

shm_unlink
IF *PCTS_shm_unlink* **THEN**
TEST: A successful call to the function *shm_unlink*() removes the name of the shared memory object named by the string pointed to by *name*, and returns 0.
ELSE *NO_OPTION*
Conformance for shm_unlink: PASS, NO_OPTION

5 **IF** *PCTS_shm_unlink* **THEN**
IF *PCTS_shm_open* **THEN**
TEST: When one or more references to the shared memory object exist when the object is unlinked, the removal of the memory object contents is postponed until all open references to the shared memory object have been removed.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for shm_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** *PCTS_shm_unlink* **THEN**
IF *PCTS_shm_open* and *PCTS_mmap* **THEN**
TEST: When one or more references to the shared memory object exist when the object is unlinked, the removal of the memory object contents is postponed until all map references to the shared memory object have been removed.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for shm_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D_1 **IF** *PCTS_shm_unlink* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *shm_unlink*() function does so in 12.3.2.2.
ELSE *NO_OPTION*
Conformance for shm_unlink: PASS, NO_OPTION

R_1 **IF** *PCTS_shm_unlink* **THEN**
TEST: When a call to *shm_unlink*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: All assertions in 12.3.2.2

R_2 **IF** *PCTS_shm_unlink* **THEN**
TEST: When a call to *shm_unlink*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error; the named shared memory object is unchanged.
ELSE *NO_OPTION*
SEE: All assertions in 12.3.2.4.

12.3.2.4 Errors

7 **IF** *PCTS_shm_unlink* and {_POSIX_NO_TRUNC} **THEN**
TEST: A call to *shm_unlink( )*, when permission is denied to unlink the named shared memory object, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for shm_unlink: PASS, NO_OPTION

8 **IF** *PCTS_shm_unlink* **THEN**
IF {POSIX_NO_TRUNC} and {NAME_MAX}<= *PCTS_NAME_MAX* **THEN**
TEST: A call to *shm_unlink( )*, when the length of the *name* string exceeds {NAME_MAX}, returns a value of -1 and sets *errno* to [ENAMETOOLONG].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for shm_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_shm_unlink* **THEN**
TEST: A call to *shm_unlink(name)*, when the named shared memory object does not exist, returns a value of -1 and sets *errno* to [ENOENT].
ELSE *NO_OPTION*
Conformance for shm_unlink: PASS, NO_OPTION

10 **IF** not *PCTS_shm_unlink* **THEN**
TEST: A call to *shm_unlink( )* returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for shm_unlink: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

--`,,`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Section 13: Execution Scheduling

13.1 Scheduling Parameters

1 **SETUP:** Include the header `<sched.h>`.
TEST: The structure *sched_param* is defined and has the member

Member Type	Member Name	Description
int	*sched_priority*	Process execution scheduling priority

Conformance for sched_param: PASS

D_1 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents extensions to *sched_param*, as permitted in POSIX.1b{3}, 1.3.1.1 item (2), does so in 13.1.
ELSE *NO_OPTION*
Conformance for sched_param: PASS, NO_OPTION

2 **SETUP:** Include the header `<sched.h>`.
TEST: Extensions to *sched_param* that may change the behavior of the application with respect to this standard when those fields in the structure are uninitialized, are enabled as required byPOSIX.1b {3} 1.3.1.1.
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.
Conformance for sched_param: PASS, NO_TEST

3 **SETUP:** Include the header `<sched.h>`.
TEST: The symbols allowed by this standard to be in the header `<time.h>` are visible.
Conformance for sched_param: PASS

13.2 Scheduling Policies

1 **TEST:** The implementation makes the process at the head of the highest priority nonempty process list a running process, regardless of its associated policy.
Conformance for sched_policy: *PASS*

2 **TEST:** A running process is then removed from its process list.
NOTE: There is no known portable test method for this assertion.
Conformance for sched_policy: PASS, NO_TEST

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D_1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the implementation of scheduling policies other SCHED_FIFO, SCHED_RR, and SCHED_OTHER, does so in 13.2.
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_OPTION

3 **SETUP:** Include the header `<sched.h>`.
TEST: The constants SCHED_FIFO, SCHED_RR, and SCHED_OTHER are defined and are bitwise distinct.
Conformance for sched_policy: PASS

13.2.1 SCHED_FIFO

sched_fifo1
IF *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Processes scheduled under the FIFO scheduling policy are chosen from a process list that is ordered by the time its processes have been on the list without being executed.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

sched_fifo2
IF *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Under the FIFO scheduling policy, when a running process becomes a preempted process, it becomes the head of the process list for its priority.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

sched_fifo3
IF *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Under the SCHED_FIFO policy, when a blocked process becomes a runnable process, it becomes the tail of the process list for its priority.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

sched_fifo4
IF *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Under the SCHED_FIFO policy, when a running process calls the *sched_setscheduler*() function, the process specified in the function call is modified to the specified policy and the priority specified by the *param* argument.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

sched_fifo5
IF *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Under the SCHED_FIFO policy, if the process whose policy and priority has been modified is a running process or is runnable, it then becomes the tail of the process list for its new priority.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

sched_fifo6
IF *PCTS_sched_setscheduler* and *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setscheduler* and *PCTS_GAP_sched_setparam* **THEN**
TEST: Under the SCHED_FIFO policy, when a running process calls the *sched_setparam*() function, the priority of the process specified in the function call is modified to the priority specified by the *param* argument.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

sched_fifo7
IF *PCTS_sched_setscheduler* and *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setscheduler* and *PCTS_GAP_sched_setparam* **THEN**
TEST: Under the SCHED_FIFO policy, if a process whose priority has been modified is a running process or is runnable, it then becomes the tail of the process list for its new priority.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

sched_fifo8
IF *PCTS_sched_setscheduler* and *PCTS_sched_yield* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Under the SCHED_FIFO policy, when a running process issues the *sched_yield*() function, the process becomes the tail of the process list for its priority.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

R_1 TEST: Under the SCHED_FIFO policy, the position of a process within the process lists is affected only by process scheduling events.
NOTE: There is no known portable test method for this assertion.
SEE: Assertions sched_fifo1, sched_fifo2, sched_fifo3, sched_fifo4, sched_fifo5, sched_fifo6, sched_fifo7, sched_fifo8 in 13.2.1.

4 **IF** *PCTS_sched_setscheduler* and *PCTS_sched_get_priority_max* and *PCTS_sched_get_priority_min* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Valid priorities under the SCHED_FIFO policy are within the range returned by the function *sched_get_priority_max*() and *sched_get_priority_min*() when SCHED_FIFO is provided as the parameter.
TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: The SCHED_FIFO policy has a priority range of at least 32 priorities.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

13.2.2 SCHED_RR

6 **IF** *PCTS_sched_setscheduler* and *PCTS_sched_rr_get_interval* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: The SCHED_RR policy is identical to the SCHED_FIFO policy, with the additional condition that when the implementation detects that a running process has been executing as a running process for a time period of the length returned by the function *sched_rr_get_interval*() or longer, the process becomes the tail of its process list, and the head of that process list is removed and made a running process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: A process under the SCHED_RR policy that is preempted, and subsequently resumes execution as a running process, completes the unexpired portion of its round-robin-interval time period.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_sched_setscheduler* and *PCTS_sched_get_priority_max* and *PCTS_sched_get_priority_min* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: For the SCHED_RR policy, valid priorities are within the range returned by the functions *sched_get_priority_max*() and *sched_get_priority_min*() when SCHED_RR is provided as the parameter.
TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: The SCHED_RR has a priority range of at least 32 priorities.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

13.2.3 SCHED_OTHER

D_2 TEST: A PCD.1b documents the behavior of the SCHED_OTHER policy, as described in the definition of scheduling policy, in 13.2.3.
Conformance for sched_policy: PASS

D_3 TEST: The PCD.1b documents the effect of scheduling processes with the SCHED_OTHER policy, as described in a system in which other processes are executing under SCHED_FIFO or SCHED_RR, in 13.2.3.
Conformance for sched_policy: PASS

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10 **IF** *PCTS_sched_setscheduler* and *PCTS_sched_get_priority_max* and *PCTS_sched_get_priority_min* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Priorities of processes executing under the SCHED_OTHER policy are restricted to the range returned by the functions *sched_get_priority_max*() and *sched_get_priority_min*() when SCHED_OTHER is provided as the parameter.
TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

13.3 Process Scheduling Functions

13.3.1 Set Scheduling Parameters

Function: *sched_setparam*().

13.3.1.1 Synopsis

1
M_GA_stdC_proto_decl(int; sched_setparam; pid_t pid, const struct sched_param *param; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sched_setparam: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(sched_setparam; sched.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sched_setparam: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; sched_setparam; sched.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sched_setparam: PASS, NO_OPTION

4
M_GA_macro_args (sched_setparam; sched.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sched_setparam: PASS, NO_OPTION

13.3.1.2 Description

sched_setparam
IF *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: A successful call to *sched_setparam*() sets the scheduling parameters of the process specified by *pid* to the values specified by the *sched_param* structure pointed to by *param*, and returns the value 0.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: Any integer within the inclusive priority range for the current scheduling policy of the process specified by *pid* is a valid value of the *sched_priority* member in the *param* structure.
TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: Higher numerical values for the priority represent higher priorities.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_sched_setparam* a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of *sched_setparam*(), if the value of *pid* is negative, does so in 13.3.1.2.
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_OPTION

7 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: When a process specified by *pid* exists, and if the calling process has permission, the scheduling parameters are set for the process whose process ID is equal to *pid*.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: When *pid* is zero, the scheduling parameters are set for the calling process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_sched_setparam* **THEN**
TEST: A PCD.1b documents the conditions under which one process has permission to change the scheduling parameters of another process in 13.3.1.2.
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_OPTION

9 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_RAP_sched_setparam* **THEN**
TEST: The requesting process must have the appropriate privilege to set its own scheduling parameters, or those of another process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D_3 **IF** *PCTS_sched_setparam* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the requesting process must have appropriate privilege to set its own scheduling parameters, or those of another process, does so in 13.3.1.2.
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_OPTION

10 **IF** *PCTS_sched_setparam* THEN
TEST: The target process, whether it is running or not running, resumes execution after all other runnable processes of equal or greater priority have been scheduled to run.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST, NO_OPTION

11 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: When the priority of the process specified by the *pid* argument is set higher than that of the lowest-priority running process, and if the specified process is ready to run, the process specified by the *pid* argument preempts a lowest priority running process.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

12 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: When the process calling *sched_setparam*() sets its own priority lower than that of one or more other nonempty process lists, then the process that is the head of the highest priority list preempts the calling process.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

D_4 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparatm* **THEN**
TEST: A PCD.1b documents the result of the current scheduling policy for the process specified by *pid* if it is not SCHED_FIFO, or SCHED_RR, or SCHED_OTHER, in 13.3.1.2.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

D_5 **IF** *PCTS_sched_setparam* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not the implementation supports the *sched_setparam*() function, does so in 13.3.1.2.
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_OPTION

13.3.1.4 Errors

13 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: A call to *sched_setparam*(), when one or more of the requested scheduling parameters is outside the range defined for the scheduling policy of the specified *pid*, returns a value of -1 and sets *errno* to [EINVAL].

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

14 **IF** not *PCTS_sched_setparam* **THEN**

TEST: A call to *sched_setparam*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for sched_setparam: PASS, NO_OPTION

15 **IF** *PCTS_sched_setparam* **THEN**

IF *PCTS_GAP_sched_setparam* **THEN**

TEST: A call to *sched_setparam*(), when the requesting process does not have permission to set the scheduling parameters for the specified process, returns a value of -1 and sets *errno* to [EPERM].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **IF** *PCTS_sched_setparam* **THEN**

IF *PCTS_RAP_sched_setparam* **THEN**

TEST: A call to *sched_setparam*(). when the requesting process does not have the appropriate privilege to invoke *sched_setparam*(), returns a value of -1 and sets *errno* to [EPERM].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

17 **IF** *PCTS_sched_setparam* **THEN**

IF *PCTS_GAP_sched_setparam* **THEN**

TEST: A call to *sched_setparam*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].

NOTE: A subroutine is recommended that returns a *pid* that doesn't correspond to any existing process.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

13.3.2 Get Scheduling Parameters

Function: *sched_getparam*().

13.3.2.1 Synopsis

1

M_GA_stdC_proto_decl(int; sched_getparam; pid_t pid, struct sched_param *param; sched.h;;;)

SEE: Assertion GA_stdC_proto_decl in 2.7.3.

Conformance for sched_getparam: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sched_getparam; sched.h;;;)

SEE: Assertion GA_commonC_int_result_decl in 2.7.3.

Conformance for sched_getparam: PASS[1, 2], NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3

M_GA_macro_result_decl(int; sched_getparam; sched.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sched_getparam: PASS, NO_OPTION

4

M_GA_macro_args (sched_getparam; sched.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sched_getparam: PASS, NO_OPTION

sched_getparam

IF *PCTS_sched_getparam* **THEN**
TEST: A successful call to *sched_getparam*() returns the scheduling parameters of a process specified by *pid* in the *sched_param* structure pointed to by *param*, and returns the value zero.
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

5 **IF** *PCTS_sched_getparam* **THEN**
TEST: When a process specified by *pid* exists, and if the calling process has permission, the scheduling parameters for the process whose process ID is equal to *pid* are returned.
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

6 **IF** *PCTS_sched_getparam* **THEN**
TEST: When *pid* is zero, the scheduling parameters for the calling process are returned.
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

D_1 **IF** *PCTS_sched_getparam* and PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of *sched_getparam*(), if the value of *pid* is negative, does so in 13.3.2.2.
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

D_2 **IF** *PCTS_sched_getparam* and PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_getparam*() function does so in 13.3.2.2.
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

R_1 **IF** *PCTS_sched_getparam* **THEN**
TEST: When a call to *sched_getparam*() completes successfully, the interface returns 0.
ELSE *NO_OPTION*
SEE: Assertion sched_getparam in 13.3.2.2.

R_2 **IF** *PCTS_sched_getparam* **THEN**
TEST: When a call to *sched_getparam*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in in 13.3.2.4.

13.3.2.4 Errors

7 **IF** not *PCTS_sched_getparam* **THEN**
TEST: A call to *sched_getparam*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8 **IF** *PCTS_sched_getparam* **THEN**
TEST: A call to *sched_getparam*(), when the requesting process does not have permission to obtain the scheduling parameters of the specified process, returns a value of -1 and sets *errno* to [EPERM].
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

9 **IF** *PCTS_sched_getparam* **THEN**
TEST: A call to *sched_getparam*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].
NOTE: A subroutine is recommended that returns a *pid* that doesn't correspond to any existing process.
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

13.3.3 Set Scheduling Policy and Scheduling Parameters

Function: *sched_setscheduler*().

1
M_GA_stdC_proto_decl(int; sched_setscheduler; pid_t pid, int policy, const struct sched_param *param; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sched_setscheduler: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(sched_setscheduler; sched.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sched_setscheduler: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; sched_setscheduler; sched.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sched_setscheduler: PASS, NO_OPTION

4
M_GA_macro_args (sched_setscheduler; sched.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sched_setscheduler: PASS, NO_OPTION

13.3.3.2 Description

sched_setscheduler
IF *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: A successful call to *sched_setscheduler*() sets the scheduling policy of the process specified by *pid* to *policy*, and its scheduling parameters to and the parameters specified in the *sched_param* structure pointed to by *param*, and returns the former scheduling policy of the specified process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: The *sched_priority* member in the *param* structure can take on any integer value within the inclusive priority range for the scheduling policy specified by *policy*.

TR: Try to return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_sched_setscheduler* and a PCD.1b documents the following THEN

TEST: A PCD.1b that documents the behavior of *sched_setscheduler(pid, policy, param)*, if the value of *pid* is negative, does so in 13.3.3.2.

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_OPTION

6 **IF** *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

SETUP: Include the header `<sched.h>`.

TEST: The possible values for the *policy* parameter, in the call *sched_setscheduler (pid, policy, param),* are defined.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: When a process specified by *pid* exists, and if the calling process has permission, *sched_setscheduler (pid, policy, param)* sets the scheduling policy for the process whose process ID is equal to *pid.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: When a process specified by *pid* exists, and if the calling process has permission, *sched_setscheduler (pid, policy, param)* sets the scheduling parameters for the process whose process ID is equal to *pid.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: When *pid* is zero, *sched_setscheduler (pid, policy, param)* sets the scheduling policy for the calling process.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

10 **IF** *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: When *pid* is zero, *sched_setscheduler(pid, policy, param)* sets the scheduling parameters for the calling process.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D_1 **IF** *PCTS_sched_setscheduler* **THEN**
TEST: The PCD.1b documents the conditions under which one process has the appropriate privilege to change the scheduling parameters of another process in 13.3.3.2.
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_OPTION

D_2 **IF** *PCTS_sched_setscheduler* THEN
TEST: The PCD.1b documents the conditions under which one process has the appropriate privilege to change the scheduling parameters of another process in 13.3.3.2.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

D_3 **IF** *PCTS_sched_setscheduler* and a PCD.1b documents the following **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: A PCD.1 that documents whether the requesting process has permission to set its own scheduling parameters, or those of another process, does so in 13.3.3.2.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

D_4 **IF** *PCTS_sched_setscheduler* and a PCD.1b documents the following **THEN**
TEST: A PCD.1 that documents restrictions that apply to the appropriate privileges required to set a process's own scheduling policy, or another process's scheduling policy, to a particular value, does so in 13.3.3.2.
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_OPTION

R_1 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: The *sched_setscheduler*() function is considered successful if it succeeds in setting the scheduling policy, and the scheduling parameters of the process specified by *pid*, to the values specified by *policy* and the structure *param*, respectively.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion sched_setscheduler in 13.3.3.2.

D_5 **IF** *PCTS_sched_setscheduler* and a PCD.1b documents the following **THEN**
TEST: A PCD.1 that documents whether or not it supports the *sched_setscheduler*() function, does so in 13.3.3.2.
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_OPTION

13.3.3.3 Returns

R_2 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: When a call to *sched_setscheduler*() completes successfully, the interface returns the former scheduling policy of the specified process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion sched_setscheduler in 13.3.3.2.

R_3 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: When a call to *sched_setscheduler*() completes unsuccessfully, the policy and scheduling parameters remain unchanged, and the interface returns a value of -1 and sets *errno* to indicate the error.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: All assertions in 13.3.3.4.

13.3.3.4 Errors

11 **IF** *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: A call to *sched_setscheduler*(), when the value of the *policy* parameter is invalid, returns a value of -1 and sets *errno* to [EINVAL].

NOTE: A subroutine is recommended that indicates either an invalid value for *policy*, or that there is no way to generate an invalid value for *policy* on the system.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

12 **IF** *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: A call to *sched_setscheduler*(), when one or more of the parameters contained in *param* is outside the valid range for the specified scheduling policy, returns a value of -1 and sets *errno* to [EINVAL].

TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_OPTION

13 **IF** not *PCTS_sched_setscheduler* **THEN**

TEST: A call to *sched_setscheduler*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_OPTION

14 **IF** *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: A call to *sched_setscheduler*(), when the requesting process does not have permission to set the scheduling parameters of the specified process, returns a value of -1 and sets *errno* to [EPERM].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: A call to *sched_setscheduler*(), when the requesting process does not have permission to set the scheduling policy of the specified process, returns a value of -1 and sets *errno* to [EPERM].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: A call to *sched_setscheduler*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

13.3.4 Get Scheduling Policy

Function: *sched_getscheduler*().

13.3.4.1 Synopsis

1
M_GA_stdC_proto_decl(int; sched_getscheduler; pid_t pid; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sched_getscheduler: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(sched_getscheduler; sched.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sched_getscheduler: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; sched_getscheduler; sched.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sched_getscheduler: PASS, NO_OPTION

4
M_GA_macro_args (sched_getscheduler; sched.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sched_getscheduler: PASS, NO_OPTION

13.3.4.2 Description

sched_getscheduler
IF *PCTS_sched_getscheduler* **THEN**
TEST: A successful call to *sched_getscheduler*() returns the scheduling policy of the process specified by *pid*.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

5 **IF** *PCTS_sched_getscheduler* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of *sched_getscheduler*(), if the value of *pid* is negative, does so in 13.3.4.2.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

6 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: The values that can be returned by *sched_getscheduler*() are defined in the header file `<sched.h>`.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: When a process specified by *pid* exists, and if the calling process has permission, the scheduling policy is returned for the process whose process ID is equal to *pid*.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

8 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: When *pid* is zero, the scheduling policy is returned for the calling process.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

D_1 **IF** *PCTS_sched_getscheduler* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_getscheduler*() function does so in 13.3.4.2.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

13.3.4.3 Returns

R_1 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: When a call to *sched_getscheduler*() completes successfully, the interface returns the former policy of the specified process.
ELSE *NO_OPTION*
SEE: Assertion sched_getscheduler in 13.3.4.2

R_2 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: When a call to *sched_getscheduler*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 13.3.4.4

13.3.4.4 Errors

9 **IF** not *PCTS_sched_getscheduler* **THEN**
TEST: A call to *sched_getscheduler*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

10 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: A call to *sched_getscheduler*(), when the requesting process does not have permission to determine the scheduling policy of the specified process, returns a value of -1 and sets *errno* to [EPERM].
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

11 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: A call to *sched_getscheduler*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].
NOTE: A subroutine is recommended that returns a *pid* that doesn't correspond to any existing process.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

13.3.5 Yield Processor

Function: *sched_yield*().

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2

1

M_GA_stdC_proto_decl(int; sched_yield; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sched_yield: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sched_yield; sched.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sched_yield: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sched_yield; sched.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sched_yield: PASS, NO_OPTION

4

M_GA_macro_args (sched_yield; sched.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sched_yield: PASS, NO_OPTION

13.3.5.2 Description

sched_yield

IF *PCTS_sched_yield* **THEN**
TEST: A successful call to *sched_yield*() forces the running process to relinquish the process until it again becomes the head of its process list, and returns the value zero.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sched_yield: PASS, NO_TEST, NO_OPTION

D_1 IF *PCTS_sched_yield* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_yield*() function does so in 13.3.5.2.
ELSE *NO_OPTION*
Conformance for sched_yield: PASS, NO_OPTION

13.3.5.3 Returns

R_1 IF *PCTS_sched_yield* **THEN**
TEST: When a call to *sched_yield*() completes successfully, the interface returns 0.
ELSE *NO_OPTION*
SEE: Assertion sched_yield in 13.3.5.2.

R_2 IF *PCTS_sched_yield* **THEN**
TEST: When a call to *sched_yield*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 13.3.5.4.

13.3.5.4 Errors

5 **IF** not *PCTS_sched_yield* **THEN**
TEST: A call to *sched_yield*() returns a value of -1 and sets *errno* to [ENOSYS]
ELSE *NO_OPTION*
Conformance for sched_yield: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.3.6 Get Scheduling Parameter Limits

Functions:
sched_get_priority_max(),
sched_get_priority_min(),
sched_rr_get_interval().

13.3.6.1 Synopsis

1

M_GA_stdC_proto_decl(int; sched_get_priority_max; int policy; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sched_get_priority_max: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sched_get_priority_max; sched.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sched_get_priority_max: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sched_get_priority_max; sched.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sched_get_priority_max: PASS, NO_OPTION

4

M_GA_macro_args (sched_get_priority_max; sched.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sched_get_priority_max: PASS, NO_OPTION

5

M_GA_stdC_proto_decl(int; sched_get_priority_min; int policy; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sched_get_priority_min: PASS[5,6], NO_OPTION

6

M_GA_commonC_int_result_decl(sched_get_priority_min; int policy; sched.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sched_get_priority_min: PASS[5, 6], NO_OPTION

7

M_GA_macro_result_decl(int; sched_get_priority_min; sched.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sched_get_priority_min: PASS, NO_OPTION

8

M_GA_macro_args (sched_get_priority_min; sched.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for sched_get_priority_min: PASS, NO_OPTION

9

M_GA_stdC_proto_decl(int; sched_rr_get_interval; pid_t pid, struct timespec *interval; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for sched_rr_get_interval: PASS[9, 10], NO_OPTION

10

M_GA_commonC_int_result_decl(sched_rr_get_interval; pid_t pid, struct timespec *interval; sched.h;;;)

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for sched_rr_get_interval: PASS[9, 10], NO_OPTION

11

M_GA_macro_result_decl(int; sched_rr_get_interval; sched.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for sched_rr_get_interval: PASS, NO_OPTION

12

M_GA_macro_args (sched_rr_get_interval; sched.h;;;)
SEE: Assertion GA_macro_args in 2.7.3
Conformance for sched_rr_get_interval: PASS, NO_OPTION

13.3.6.2 Description

sched_get_priority_max

IF *PCTS_sched_get_priority_max* **THEN**
TEST: A successful call to *sched_get_priority_max*() returns the appropriate maximum for the scheduling policy specified by *policy*.
TR: Test for SCHED_FIFO, SCHED_RR, and SCHED_OTHER.
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

sched_get_priority_min

IF *PCTS_sched_get_priority_min* **THEN**
TEST: A successful call to *sched_get_priority_min* () returns the appropriate minimum for the scheduling policy specified by *policy*.
TR: Test for SCHED_FIFO, SCHED_RR, and SCHED_OTHER.
ELSE *NO_OPTION*
Conformance for sched_get_priority_min: PASS, NO_OPTION

sched_rr_get_interval

IF *PCTS_sched_rr_get_interval* **THEN**
TEST: A successful call to *sched_rr_get_interval*() updates the *timespec* structure referenced by the *interval* argument to contain the current execution time limit (i.e., time quantum) for the process specified by *pid*, and returns the value zero.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sched_rr_get_interval: PASS, NO_TEST, NO_OPTION

13 **IF** *PCTS_sched_rr_ get_interval* **THEN**
TEST: When *pid* is zero, *sched_rr_get_interval*() returns the current execution time limit for the calling process.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sched_rr_get_interval: PASS, NO_TEST, NO_OPTION

14 **IF** *PCTS_sched_get_priority_max* **THEN**
TEST: In the call to *sched_get_priority_max(policy),* the value of *policy* is one of the scheduling policy values defined in `<sched.h>`.
NOTE: A subroutine is recommended that either returns an invalid scheduling policy, or indicates that there is no way to generate an invalid scheduling policy on the system.
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

15 **IF** *PCTS_sched_get_priority_min* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: In the call *sched_get_priority_min(policy),* the value of *policy* is one of the scheduling policy values defined in `<sched.h>`.

NOTE: A subroutine is recommended that either returns an invalid scheduling policy, or indicates that there is no way to generate an invalid scheduling policy on the system.

ELSE *NO_OPTION*
Conformance for sched_get_priority_min: PASS, NO_OPTION

D_1 IF *PCTS_sched_get_priority_max* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_get_priority_max*() function does so in 13.3.6.2.
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

D_2 IF *PCTS_sched_get_priority_min* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_get_priority_min*() function does so in 13.3.6.2.
ELSE *NO_OPTION*
Conformance for sched_get_priority_min: PASS, NO_OPTION

D_3 IF *PCTS_sched_rr_get_interval* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_rr_get_interval*() function does so in 13.3.6.2.
ELSE *NO_OPTION*
Conformance for sched_rr_get_interval: PASS, NO_OPTION

13.3.6.3 Returns

R_1 IF *PCTS_sched_get_priority_max* **THEN**
TEST: When a call to *sched_get_priority_max*() completes successfully, the interface returns the appropriate maximum value.
ELSE *NO_OPTION*
SEE: Assertion sched_get_priority_max in 13.3.6.2.

R_2 IF *PCTS_sched_get_priority_min* **THEN**
TEST: When a call to *sched_get_priority_min*() completes successfully, the interface returns the appropriate minimum value.
ELSE *NO_OPTION*
SEE: Assertion sched_get_priority_min in 13.3.6.2.

R_3 IF *PCTS_sched_get_priority_max* **THEN**
TEST: When a call to *sched_get_priority_max*() completes successfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 13.3.6.4.

R_4 IF *PCTS_sched_get_priority_min* **THEN**
TEST: When a call to *sched_get_priority_min*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 13.3.6.4.

R_5 IF *PCTS_sched_rr_get_interval* **THEN**
TEST: When a call to *sched_rr_get_interval*(, completes successfully, the interface returns 0.
ELSE *NO_OPTION*
SEE: Assertion sched_rr_get_interval in 13.3.6.2.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

R_6 IF *PCTS_sched_rr_get_interval* **THEN**
TEST: When a call to *sched_rr_get_interval*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 13.3.6.4.

13.3.6.4 Errors

14 **IF** *PCTS_sched_get_priority_max* **THEN**
TEST: A call to *sched_get_priority_max*(), when the value of the *policy* parameter does not represent a defined scheduling policy, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid scheduling policy, or indicates that there is no way to generate an invalid scheduling policy on the system.
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

17 **IF** *PCTS_sched_get_priority_min* **THEN**
TEST: A call to *sched_get_priority_min*(), when the value of the *policy* parameter does not represent a defined scheduling policy, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid scheduling policy, or indicates that there is no way to generate an invalid scheduling policy on the system.
ELSE *NO_OPTION*
Conformance for sched_get_priority_min: PASS, NO_OPTION

18 **IF** not *PCTS_sched_get_priority_max* **THEN**
TEST: A call to *sched_get_priority_max*() returns a value of -1 and sets *errno* to [ENOSYS]
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

19 **IF** not *PCTS_sched_get_priority_min* **THEN**
TEST: A call to *sched_get_priority_min*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sched_get_priority_min: PASS, NO_OPTION

20 **IF** not *PCTS_sched_rr_get_interval* **THEN**
TEST: A call to *sched_rr_get_interval*()returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sched_rr_get_interval: PASS, NO_OPTION

21 **IF** *PCTS_sched_get_priority_max* **THEN**
TEST: A call to *sched_get_priority_max*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

22 **IF** *PCTS_sched_get_priority_min* **THEN**
TEST: A call to *sched_get_priority_min*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].
NOTE: A subroutine is recommended that returns a *pid* that doesn't correspond to any existing process.
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for sched_get_priority_min: PASS, NO_OPTION

23 **IF** *PCTS_sched_rr_get_interval* **THEN**

TEST: A call to *sched_rr_get_interval*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].

NOTE: A subroutine is recommended that returns a *pid* that doesn't correspond to any existing process.

ELSE *NO_OPTION*

Conformance for sched_rr_get_interval: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Section 14: Clocks and Timers

14.1 Data Definitions for Clocks and Timers

14.1.1 Time Value Specification Structures

1 **SETUP:** Include the header `<time.h>`.
TEST: The structure *timespec* is defined, and includes the members

Member Type	Member Name	Description
time_t	*tv_sec*	Seconds
long	*tv_nsec*	Nanoseconds

Conformance for timer_hdr: PASS

D_1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents extensions to *timespec,* as permitted in POSIX.1b{3}, 1.3.1.1 item (2), does so in 14.1.1.
ELSE *NO_OPTION*
Conformance for timer_hdr: PASS, NO_OPTION

2 **SETUP:** Include the header `<time.h>`.
TEST: Extensions to *timespec* that may change the behavior of the application with respect to this standard when those fields in the structure are uninitialized, are enabled as required byPOSIX.1b {3} 1.3.1.1.
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.
Conformance for timer_hdr: PASS, NO_TEST

3 **TEST:** The *tv_nsec* member of *timespec* is valid only if it is less than the number of nanoseconds in a second (1000 million).
Conformance for timer_hdr: PASS

4 **IF** *{int_max}* <=10E9 **THEN**
TEST: The *tv_nsec* member of *timespec* is valid only if it less than the number of nanoseconds in a second (1000 million).
ELSE *NO_TEST_SUPPORT*
Conformance for timer_hdr: PASS, NO_TEST_SUPPORT

5 **TEST:** The time interval described by *timespec* is *(tv_sec* x 10^9 + *tv_nsec*) nanoseconds.
Conformance for timer_hdr: PASS

6 **SETUP:** Include the header `<time.h>`.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: The structure *itimerspec* is defined and includes the members

Member Type	Member Name	Description
struct timespec	*it_interval*	Timer period
struct timespec	*it_value*	Timer expiration

Conformance for timer_hdr: PASS

D_2 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents extensions to *itimerspec*, as permitted in POSIX1b{3}, 1.3.1.1, item (2), does so in 14.1.
Conformance for timer_hdr:PASS, NO_OPTION

7 **SETUP:** Include the header `<time.h>`.
TEST: Extensions to *itimerspec* that may change the behavior of the application with respect to this standard, when those fields in the structure are uninitialized, are enabled as required by POSIX.1b {3} 1.3.1.1.
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.
Conformance for timer_hdr: PASS, NO_TEST

8 **TEST:** When the value described by *it_value* is nonzero, it indicates the time to or time of the next timer expiration for relative and absolute timer values, respectively.
Conformance for timer_hdr: PASS

9 **TEST:** When the value described by *it_value* is zero, the timer is disarmed.
Conformance for timer_hdr: PASS

10 **TEST:** When the value described by *it_interval* is nonzero, it specifies an interval to be used in reloading the timer when it expires; that is, a periodic timer is specified.
Conformance for timer_hdr: PASS

11 **TEST:** When the value described by *it_interval* is zero, the timer is disarmed after its next expiration; that is, a "one-shot" timer is specified.
Conformance for timer_hdr: PASS

14.1.2 Timer Event Notification Control Block

12 **IF** {_POSIX_REALTIME_SIGNALS}**THEN**
TEST: Per-process timers can be created that notify the process of timer expirations by queuing a realtime extended signal.
ELSE *NO_TEST_SUPPORT*
Conformance for timer_hdr: PASS, NO_TEST_SUPPORT

14.1.3 Type Definitions

13 **SETUP:** Include the header `<sys/types.h>`.
TEST: The types *clockid_t* and *timer_t* are defined.
Conformance for timer_hdr: PASS

14.1.4 Manifest Constants

14 **SETUP:** Include the header `<time.h>`.
TEST: The constants CLOCK_REALTIME and TIMER_ABSTIME are defined.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance for timer_hdr: PASS

15 **SETUP:** Include the header <unistd.h>.
TEST: The constant {_POSIX_CLOCKRES_MIN} is defined as 20 ms (1/50 of a second).
Conformance for timer_hdr: PASS

16 **TEST:** The maximum allowable resolution for the CLOCK_REALTIME clock, and all times based on this clock, including the *nanosleep*() function, is {_POSIX_CLOCKRES_MIN}.
Conformance for timer_hdr: PASS

D_3 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents support for smaller values of resolution for the CLOCK_REALTIME clock to provide finer granularity time bases, does so in 14.1.4.
ELSE *NO_OPTION*
Conformance for timer_hdr: PASS, NO_OPTION

17 **TEST:** The actual resolution for a specific clock is obtained using functions defined in this section.
Conformance for timer_hdr: PASS

D_4 IF a PCD.1b documents the following **THEN**
TEST: The PCD.1b documents the actual resolution supported for the *nanosleep*() function of timers based on this clock, if it differs from the resolution supported for the clock, in 14.1.4.
ELSE *NO_OPTION*
Conformance for timer_hdr: PASS, NO_OPTION

18 **TEST:** The maximum value of the CLOCK_REALTIME clock, and of the absolute timers based on it, is at least the same as that defined by the C Standard {2} for the *time_t* type.
Conformance for timer_hdr: PASS

D_5 IF a PCD.1b documents the following **THEN**
TEST: The PCD.1b documents the difference between the maximum value supported by the *nanosleep*() function or timers based on this clock and the maximum value supported by the clock, if such a difference exists, in 14.1.4.
Conformance for timer_hdr: PASS, NO_OPTION

14.2 Clocks and Timer Functions

14.2.1 Clocks

Functions: *clock_settime*(), *clock_gettime*(), *clock_getres*().

14.2.1.1 Synopsis

1

M_GA_stdC_proto_decl(int; clock_settime; clockid_t clock_id, const struct timespec *tp; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for clock_settime: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(clock_settime; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for clock_settime: PASS[1, 2], NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3

M_GA_macro_result_decl(int; clock_settime; time.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for clock_settime: PASS, NO_OPTION

4

M_GA_macro_args (clock_settime; time.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for clock_settime: PASS, NO_OPTION

5

M_GA_stdC_proto_decl(int; clock_gettime; clockid_t clock_id, struct timespec *tp; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for clock_gettime: PASS[5, 6], NO_OPTION

6

M_GA_commonC_int_result_decl(clock_gettime; clockid_t, clock_id, struct timespec *tp; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for clock_gettime: PASS[5, 6], NO_OPTION

7

M_GA_macro_result_decl(int; clock_gettime; time.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for clock_gettime: PASS, NO_OPTION

8

M_GA_macro_args (clock_gettime; time.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for clock_gettime: PASS, NO_OPTION

9

M_GA_stdC_proto_decl(int; clock_getres; , clockid_t clock_id, struct timespec *res; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3
Conformance for clock_getres: PASS[9, 10], NO_OPTION

10

M_GA_commonC_int_result_decl(clock_getres; , clockid_t clock_id, struct timespec *res; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3
Conformance for clock_getres: PASS[9, 10], NO_OPTION

11

M_GA_macro_result_decl(int; clock_getres; time.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4
Conformance for clock_getres: PASS, NO_OPTION

12

M_GA_macro_args (clock_getres; time.h;;;)
SEE: Assertion GA_macro_args in 2.7.3
Conformance for clock_getres: PASS, NO_OPTION

14.2.1.2 Description

clock_settime

IF *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: A successful call to *clock_settime (clock_id, tp)* sets the specified clock, *clock_id,* to the value specified by *tp* and returns 0.
ELSE *NO_TEST_SUPPORT*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: In a call to *clock_settime*(), time values that are between two consecutive non-negative integer multiples of the resolution of the specified clock are truncated down to the smaller multiple of the resolution.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

clock_gettime
IF *PCTS_clock_gettime* **THEN**
TEST: A successful call to *clock_gettime (clock_id, tp)* returns the current value *tp* for the specified clock, *clock_id*.
ELSE *NO_OPTION*
Conformance for clock_gettime: PASS, NO_OPTION

clock_getres
IF *PCTS_clock_getres* **THEN**
TEST: A successful call to *clock_getres (clock_id, res)*, with a non_**NULL** value of the argument *res*, stores the resolution of the clock specified by *clock_id* into the location pointed to by *res*, and returns 0.
ELSE *NO_OPTION*
Conformance for clock_getres: PASS, NO_OPTION

D_1 **IF** *PCTS_clock_getres* **THEN**
TEST: The PCD.1b documents clock resolutions, in 14.2.1.2.
ELSE *NO_OPTION*
Conformance for clock_getres: PASS, NO_OPTION

15 **TEST:** Clock resolutions cannot be set by a process.
Conformance for clock_settime: PASS

16 **IF** *PCTS_clock_getres* **THEN**
TEST: When the *res* argument to the call *clock_getres (clockid_t clock_id, res)* is **NULL**, the clock resolution is not returned.
ELSE *NO_OPTION*
Conformance for clock_getres: PASS, NO_OPTION

17 **IF** *PCTS_clock_settime* **THEN**
IF *PCTS__clock_getres* and *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: When the time argument of *clock_settime*() is not a multiple of the clock resolution *res*, as determined by a call to *clock_getres(clock_id, res)*, then the value is truncated to a multiple of *res*.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **TEST:** The PCD.1b documents whether the clocks are systemwide (i.e., visible to all processes), or per-process (i.e., measuring time that is meaningful only within a process), in 14.2.1.2.
Conformance for clock_settime: PASS

18 **SETUP:** Include the header `<time.h>`.
TEST: The constant CLOCK_REALTIME is defined.
Conformance for clock_settime: PASS

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

19 **TEST:** When the value CLOCK_REALTIME is used for *clock_id*, the call *clock_settime(clock_id, tp)* sets the realtime clock for the system.
Conformance for clock_settime: PASS

20 **TEST:** When the value CLOCK_REALTIME is used for *clock_id*, the call *clock_gettime(clock_id, tp)* interrogates the realtime clock for the system.
Conformance for clock_settime: PASS

21 **TEST:** When the value CLOCK_REALTIME is used for *clock_id*, the call *clock_get_res (, clock_id, res)* gets the resolution of the realtime clock for the system.
Conformance for clock_settime: PASS

22 **IF** *PCTS_clock_gettime* **THEN**
TEST: When the value CLOCK_REALTIME is used for *clock_id*, the value returned by *clock_gettime(clock_id, tp)* represents the amount of time (in seconds and nanoseconds) since the Epoch.
ELSE *NO_OPTION*
Conformance for clock_gettime: PASS, NO_OPTION

23 **IF** *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: When the value CLOCK_REALTIME is used for *clock_id*, the value specified by *clock_settime(clock_id, tp)* represents the amount of time (in seconds and nanoseconds) since the Epoch.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether clocks in addition to the realtime clock are supported, and the interpretation of time values for any such clocks, does so in 14.2.1.2.
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_OPTION

24 **IF** *PCTS_clock_settime* **THEN**
TEST: The PCD.1b documents the effect of setting a clock via *clock_settime*() on armed per-process times associated with that clock, in 14.2.1.2.
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_OPTION

25 **IF** *PCTS_clock_settime* **THEN**
TEST: The PCD.1b documents the appropriate privilege to set a particular clock in 14.2.1.2.
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_OPTION

D_3 **IF** *PCTS_clock_settime* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *clock_settime*() function does so in 14.2.1.2.
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_OPTION

D_4 **IF** *PCTS_clock_gettime* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *clock_gettime*() function does so in 14.2.1.2.
ELSE *NO_OPTION*
Conformance for clock_gettime: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D_5 IF *PCTS_clock_getres* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *clock_getres*() function does so in 14.2.1.2.
ELSE *NO_OPTION*
Conformance for clock_getres: PASS, NO_OPTION

14.2.1.3 Returns

R_1 IF *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: When a call to *clock_settime*() completes successfully, the interface returns a value of 0.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion clock_settime in 14.2.1.4.

R_2 IF *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: When a call to *clock_settime*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: All assertions in 14.2.1.4 controlled by *clock_settime*().

R_3 IF *PCTS_clock_gettime* THEN
TEST: When a call to *clock_gettime*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion clock_gettime in 14.2.1.4.

R_4 IF *PCTS_clock_gettime* THEN
TEST: When a call to *clock_gettime*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.
clock_gettime()
ELSE *NO_OPTION*
SEE: All assertions in 14.2.1.4 controlled by *clock_gettime*().

R_5 IF *PCTS_clock_getres* **THEN**
TEST: When a call to *clock_getres*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion clock_getres in 14.2.1.4.

R_6 IF *PCTS_clock_getres* **THEN**
TEST: When a call to *clock_gettime*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 14.2.1.4 controlled by *clock_getres*().

14.2.1.4 Errors

26 IF *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: A call to *clock_settime (clock_id, tp),* when the *clock_id* argument does not specify a known clock, returns a value of -1 and sets *errno* to [EINVAL].

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE: A subroutine is recommended that either returns an invalid clock name, or indicates that there is no way to generate an invalid clock name on the system.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

27 **IF** *PCTS_clock_gettime* THEN

TEST: A call to *clock_gettime (, clock_id, tp),* when the *clock_id* argument does not specify a known clock, returns a value of -1 and sets *errno* to [EINVAL].

NOTE: A subroutine is recommended that either returns an invalid clock name, or indicates that there is no way to generate an invalid clock name on the system.

ELSE *NO_OPTION*

Conformance for clock_gettime: PASS, NO_OPTION

28 **IF** *PCTS_clock_getres* **THEN**

TEST: A call to *clock_getres (, clock_id, res),* when the *clock_id* argument does not specify a known clock, returns a value of -1 and sets *errno* to [EINVAL].

NOTE: A subroutine is recommended that either returns an invalid clock name, or indicates that there is no way to generate an invalid clock name on the system.

ELSE *NO_OPTION*

Conformance for clock_getres: PASS, NO_OPTION

29 **IF** not *PCTS_clock_gettime* **THEN**

TEST: A call to *clock_settime*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for clock_settime: PASS, NO_OPTION

30 **IF** not *PCTS_clock_gettime* **THEN**

TEST: A call to *clock_gettime*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for clock_gettime: PASS, NO_OPTION

31 **IF** not *PCTS_clock_getres* **THEN**

TEST: A call to *clock_getres*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for clock_getres: PASS, NO_OPTION

32 **IF** *PCTS_clock_settime* **THEN**

IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**

TEST: A call to *clock_settime (clock_id, tp),* when the *tp* argument is outside the range for te given clock id, returns a value of -1 and sets *errno* to [EINVAL].

NOTE: A subroutine is recommended that either returns a value outside the range value for any given clock id, or indicates that there is no way to generate a value outside the range value for any given clock id on the system.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

33 **IF** *PCTS_clock_settime* **THEN**

IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**

TEST: A call to *clock_settime (clock_id, tp),* when the *tp* argument specified a nanosecond value less than zero, returns a value of -1 and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

34 **IF** *PCTS_clock_settime* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* and {INT_MAX} <=10e9 **THEN**

TEST: A call to *clock_settime (clock_id, tp),* when the *tp* argument specified a nanosecond value greater than or equal to 1000 million, returns a value of -1 and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

35 **IF** *PCTS_clock_settime* **THEN**

IF *PCTS_RAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**

TEST: A call to *clock_settime*(), when the requesting process does not have the appropriate privilege to set the specified clock, returns a value of -1 and sets *errno* to [EPERM].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

14.2.2 Create a Per-Process Timer

Function: *timer_create*().

14.2.2.1 Synopsis

1

M_GA_stdC_proto_decl(int; timer_create; clockid_t clock_id, struct sigevent *evp, timer_t *timerid; signal.h; time.h;;)

SEE: Assertion GA_stdC_proto_decl in 2.7.3.

Conformance for timer_create: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(timer_create; signal.h; time.h;;)

SEE: Assertion GA_commonC_int_result_decl in 2.7.3.

Conformance for timer_create: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; timer_create; signal.h; time.h;;)

SEE: Assertion GA_macro_result_decl in 1.3.4.

Conformance for timer_create: PASS, NO_OPTION

4

M_GA_macro_args (timer_create; signal.h; time.h;;)

SEE: Assertion GA_macro_args in 2.7.3.

Conformance for timer_create: PASS, NO_OPTION

14.2.2.2 Description

timer_create

IF *PCTS_timer_create* **THEN**

TEST: A successful call to *timer_create(clock_id, evp, timerid)* creates a per-process timer using the specified clock *clock_id* as the timing base; sets the location referenced by *timerid* to a timer ID of type *timer_t*, used to identify the timer in timer requests (see 14.2.4), and returns the value zero.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

5 **IF** *PCTS_timer_create* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TEST: Following a call to *timer_create(clock_id, evp, timerid),* the timer ID, to which the argument *timerid* is set, is unique within the calling process until the timer is deleted.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

6 **TEST:** In calls to *timer_create(clock_id, evp, timerid),* all legal values of *clock_id* are defined in `<time.h>`.

Conformance for timer_create: PASS

7 **IF** *PCTS_timer_create* **THEN**

TEST: The timer whose ID is returned is in a disarmed state upon return from *timer_create*().

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

8 **IF** *PCTS_timer_create* **THEN**

TEST: Following the call *timer_create(clock_id, evp, timerid*), the *evp* argument, if non-**NULL**, points to a *sigevent* structure that defines the asynchronous notification that will occur when the timer expires.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

9 **IF** *PCTS_timer_create* **THEN**

TEST: In the call *timer_create(clock_id, evp, timerid),* the *evp* argument must be allocated by the application.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

10 **IF** *PCTS_timer_create* **THEN**

SETUP: Include the header `<time.h>`.

TEST: The constants SIGEV_SIGNAL and SIGEV_NONE are defined and have different values.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

11 **IF** *PCTS_timer_create* **THEN**

TEST: Following the call *timer_create(clock_id, evp, timerid),* if the *sigev_notify* member of *evp* is SIGEV_SIGNAL, then the structure contains the signal number and an application-specific data value to be sent to the process when the timer expires.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

12 **IF** *PCTS_timer_create* **THEN**

TEST: Following the call *timer_create(clock_id, evp, timerid),* if the *sigev_notify* member is SIGEV_NONE, no notification is sent.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

13 **IF** *PCTS_timer_create* **THEN**

TEST: The PCD.1b documents the behavior for any value of *sigev_notify*, other than SIGEV_NONE or SIGEV_SIGNAL, in 14.2.2.2.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

14 **IF** *PCTS_timer_create* **THEN**

TEST: There is a set of clocks that can be used as timing bases for per-process timers.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

15 **IF** *PCTS_timer_create* **THEN**
TEST: There is at least one mechanism for notifying the process of timer expiration events.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

D_1 **IF** *PCTS_timer_create* **THEN**
TEST: The PCD.1b documents the set of clocks that can be used as timing bases for per-process timers, and one or more mechanisms for notifying the process of timer expiration events, in 14.2.2.2.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

16 **IF** *PCTS_timer_create* **THEN**
SETUP: Include the header `<time.h>`.
TEST: The constant CLOCK_REALTIME is defined.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

17 **IF** *PCTS_timer_create* **THEN**
TEST: In a call to *timer_create(clock_id, evp, timerid),* CLOCK_REALTIME is a legitimate value for *clock_id.*
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

18 **IF** *PCTS_timer_create* **THEN**
IF {_POSIX_REALTIME_SIGNALS} **THEN**
SETUP: Call *timer_create(clock_id, evp, timerid),* with *evp* argument pointing to a *sigevent* structure that specifies SIGEV_SIGNAL, and SA_SIGINFO set for the expiration signal.
TEST: The signal and application-defined value specified in the *sigevent* structure are queued to the process on timer expiration.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_timer_create* **THEN**
IF {_POSIX_REALTIME_SIGNALS} **THEN**
SETUP: Call *timer_create(clock_id, evp, timerid),* with *evp* argument set to **NULL** and SA_SIGINFO set for the expiration signal.
TEST: A default signal is queued to the process, and the signal data value is set to the timer ID.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_timer_create* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the realtime signal is queued and what value, if any, is sent, if SA_SIGINFO is not set for the expiration signal, does so in 14.2.2.2.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

20 **IF** *PCTS_timer_create* **THEN**
IF not {_POSIX_REALTIME_SIGNALS} **THEN**
SETUP: Call *timer_create(clock_id, evp, timerid),* with *evp* pointing to a *sigevent* structure that specifies SIGEV_SIGNAL.

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: The signal number defined in the *sigevent* structure is sent to the process on timer expiration.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **IF** *PCTS_timer_create* **THEN**

IF not {_POSIX_REALTIME_SIGNALS} **THEN**

TEST: Following the call *timer_create(clock_id, evp, timerid),* if *evp* is **NULL**, then a default signal is sent to the process.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_TEST_SUPPORT, NO_OPTION

22 **IF** *PCTS_timer_create* **THEN**

SETUP: Call *timer_create(clock_id, evp, timerid),* with a *clock_id* value of CLOCK_REALTIME and an *evp* of **NULL.**

TEST: The default signal is SIGALRM.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

23 **IF** *PCTS_timer_create* **THEN**

TEST: The PCD.1b documents the default signal number for any clock other than CLOCK_REALTIME in 14.2.2.2.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

24 **IF** *PCTS_timer_create* **THEN**

TEST: Per-process timers are not inherited by a child process across a *fork*().

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

25 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *exelp*(), and *execvp*()

IF *PCTS_timer_create* **THEN**

TEST: Per-process timers are disarmed and deleted by a successful call to *function*().

NOTE: The assertion is to be tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

D_3 **IF** *PCTS_timer_create* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *timer_create*() function does so in 14.2.2.2.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

14.2.2.3 Returns

R_1 **IF** *PCTS_timer_create* **THEN**

TEST: When a call to *timer_create(clock_id, evp, timerid)* completes successfully, the interface returns a value of 0, and updates the location referenced by *timerid* to a *timer_t*, which can be passed to the per-process timer calls (see 14.2.4).

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ELSE *NO_OPTION*
SEE: Assertion timer_create in 14.2.2.4.

R_2 IF *PCTS_timer_create* **THEN**
TEST: When a call to *timer_create(clock_id, evp, timerid)* completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 14.2.2.4.

D_4 IF *PCTS_timer_create* and A PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the value of *timerid* if an error occurs, following a call to *timer_create(clock_id, evp, timerid*), does so in 14.2.2.4.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

14.2.2.4 Errors

26 **IF** *PCTS_timer_create* **THEN**
TEST: A call to *timer_create*(), when the system lacks sufficient signal queuing resources to honor the request, returns a value of -1 and sets *errno* to [EAGAIN].
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

27 **IF** *PCTS_timer_create* **THEN**
IF {TIMER_MAX} <= *PCTS_TIMER_MAX* **THEN**
TEST: A call to *timer_create*(), when the calling process has already created all of the timers it is allowed by this implementation, returns a value of -1 and sets *errno* to [EAGAIN].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_TEST_SUPPORT, NO_OPTION

28 **IF** *PCTS_timer_create* **THEN**
TEST: A call to *timer_create*(), when the specified clock ID is not defined, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns a value outside the range value for any given clock id, or indicates that there is no way to generate a value outside the range value for any given clock id on the system.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

29 **IF** not *PCTS_timer_create* **THEN**
TEST: A call to *timer_create*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

14.2.3 Delete a Per-Process Timer

Function: *timer_delete*().

14.2.3.1 Synopsis

1
M_GA_stdC_proto_decl(int; timer_delete; timer_t timerid; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for timer_delete: PASS[1, 2], NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2

M_GA_commonC_int_result_decl(timer_delete; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for timer_delete: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; timer_delete; time.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for timer_delete: PASS, NO_OPTION

4

M_GA_macro_args (timer_delete; time.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for timer_delete: PASS, NO_OPTION

14.2.3.2 Description

timer_delete

IF *PCTS_timer_delete* **THEN**
IF *PCTS_timer_create* **THEN**
TEST: A successful call to *timer_delete(timerid)* deletes the specified timer *timerid*, previously created by the *timer_create*() function, and returns the value zero.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_timer_delete* **THEN**
TEST: When the timer is armed when *timer_delete*() is called, the behavior is as if the timer is automatically disarmed before removal.
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_OPTION

D_1 **IF** *PCTS_timer_delete* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the disposition of pending signals for the deleted timer, does so in 14.2.3.2.
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_OPTION

D_2 **IF** *PCTS_timer_delete* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *timer_delete*() function does so in 14.2.3.2.
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_OPTION

14.2.3.3 Returns

R_1 **IF** *PCTS_timer_delete* **THEN**
TEST: When a call to *timer_delete*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion timer_delete in 14.2.3.4.

R_2 **IF** *PCTS_timer_delete* **THEN**
TEST: When a call to *timer_delete*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 14.2.3.4.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.2.3.4 Errors

6 **IF** *PCTS_timer_delete* **THEN**
TEST: A call to *timer_delete(timerid)*, when the timer ID specified by *timerid* is not a valid timer ID, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid timer ID, or indicates that there is no way to generate an invalid timer ID on the system.
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_OPTION

7 **IF** not *PCTS_timer_delete* **THEN**
TEST: A call to *timer_delete*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_OPTION

14.2.4 Per-Process Timers

Functions: *timer_settime*(), *timer_gettime*(), *timer_getoverrun*()

14.2.4.1 Synopsis

1
M_GA_stdC_proto_decl(int; timer_settime; timer_t timerid, int flags, const struct itimerspec *value, struct itimerspec *ovalue; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for timer_settime: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(timer_settime; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for timer_settime: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; timer_settime; time.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for timer_settime: PASS, NO_OPTION

4
M_GA_macro_args (timer_settime; time.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for timer_settime: PASS, NO_OPTION

5
M_GA_stdC_proto_decl(int; timer_gettime; timer_t timerid, struct itimerspec *value; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for timer_gettime: PASS[5, 6], NO_OPTION

6
M_GA_commonC_int_result_decl(timer_gettime; timer_t timerid, struct itimerspec *value; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for timer_gettime: PASS[5, 6], NO_OPTION

7
M_GA_macro_result_decl(int; timer_gettime; time.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for timer_gettime: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8

M_GA_macro_args (timer_gettime; time.h;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for timer_gettime: PASS, NO_OPTION

9

M_GA_stdC_proto_decl(int; timer_getoverrun; , timer_t timerid; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for timer_getoverrun: PASS[9, 10], NO_OPTION

10

M_GA_commonC_int_result_decl(timer_getoverrun;,timer_t timerid; time.h;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for timer_getoverrun: PASS[9, 10], NO_OPTION

11

M_GA_macro_result_decl(int; timer_getoverrun; time.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for timer_getoverrun: PASS, NO_OPTION

12

M_GA_macro_args (timer_getoverrun; time.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for timer_getoverrun: PASS, NO_OPTION

14.2.4.2 Description

timer_settime

IF *PCTS_timer_settime* **THEN**
TEST: A successful call to *timer_settime(timerid, flags, value, ovalue)* sets the time until the next expiration of the timer specified by *timerid* from the *it_value* member of the *value* is nonzero, and returns the value zero.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

13 **IF** *PCTS_timer_settime* **THEN**
TEST: When the specified timer is already armed when *timer_settime(timerid, flags, value, ovalue)* is called, this call resets the time until the next expiration to the *value* specified.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

14 **IF** *PCTS_timer_settime* **THEN**
TEST: In a successful call to *timer_settime(timerid, flags, value, ovalue)*, if the *it_value* member of *value* is zero, the timer is disarmed.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

D_1 **IF** *PCTS_timer_settime* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect of disarming or resetting a timer on pending expiration notifications, does so in 14.2.4.2.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

15 **IF** *PCTS_timer_settime* **THEN**
SETUP: Include the header `<timer.h>`.
TEST: The constant TIMER_ABSTIME is defined.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16 **IF** *PCTS_timer_settime* **THEN**

TEST: When the flag TIMER_ABSTIME is not set in the argument *flags*, *timer_settime(timerid, flags, value, ovalue)* behaves as if the time until next expiration is set to be equal to the interval specified by the *it_value* member of *value*; that is, the timer expires in *it_value* nanoseconds from when the call is made (see POSIX.1b {3}, 14.1.1).

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_OPTION

17 **IF** *PCTS_timer_settime* **THEN**

TEST: When the flag TIMER_ABSTIME is not set in the argument *flags*, *timer_settime(timerid, flags, value, ovalue)* behaves as if the time until the next expiration is set to be equal to the difference between the absolute time specified by the *it_value* member of *value* and the current value of the clock associated with *timerid*; that is, the timer expires when the clock reaches the value specified by the *it_value* member of *value*.

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_OPTION

18 **IF** *PCTS_timer_settime* **THEN**

TEST: When the flag TIMER_ABSTIME is set in the argument *flags*, and the specified time has already passed, *timer_settime(timerid, flags, value, ovalue)* succeeds and the expiration notification is made.

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_OPTION

19 **IF** *PCTS_timer_settime* **THEN**

TEST: A successful call to *timer_settime(timerid, flags, value, ovalue*), with a zero value of *it_interval,* specifies a nonperiodic (one-time) timer.

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_OPTION

20 **IF** *PCTS_timer_settime* **THEN**

TEST: A successful call to *timer_settime(timerid, flags, value, ovalue*), with a nonzero *it_interval* specifies a periodic (or repetitive) timer, with the reload value of the timer set to the value specified by the *it_interval* member of *value*.

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_OPTION

21 **IF** *PCTS_timer_settime* **THEN**

TEST: In calls to *timer_settime*(), time values that are between two consecutive nonnegative integer multiples of the resolution of the specified timer are founded up to the larger multiple of the resolution; quantization error does not cause the timer to expire earlier than the rounded time value.

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_OPTION

22 **IF** *PCTS_timer_settime* **THEN**

TEST: When the timer *timerid* is armed and the argument *ovalue* is not **NULL**, the call *timer_settime(timerid, flags, value, ovalue)* stores, in the location referenced by *ovalue,* a value representing the previous amount of time before the timer would have expired and the previous timer reload value.

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_OPTION

23 **IF** *PCTS_timer_settime* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: When the timer *timerid* is disarmed and the argument *ovalue* is not **NULL**, the call *timer_settime(timerid, flags, value, ovalue)* stores, in the location referenced by *ovalue,* a value representing zero and the previous timer reload value.

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_OPTION

24 **IF** *PCTS_timer_settime* **THEN**

IF *PCTS_timer_gettime* **THEN**

TEST: In the call *timer_settime(timerid, flags, value, ovalue)*, the members of *ovalue* are subject to the resolution of the timer, and they are the same values that would be returned by a *timer_gettime*() call at that point in the time.

NOTE: There is no known portable test method for this assertion.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

timer_gettime

IF *PCTS_timer_gettime* **THEN**

TEST: A successful call to *timer_gettime*() stores both the reload value of the timer and the amount of time until the specified timer *timerid* expires, into the space pointed to by the *value* argument; and returns the value zero.

The *it_value* member of this structure contains the amount of time before the timer expires, or zero if the timer is disarmed.

The *it_interval* member of *value* contains the reload value last set by *timer_settime*().

ELSE *NO_OPTION*

Conformance for timer_gettime: PASS, NO_OPTION

25 **IF** *PCTS_timer_gettime* **THEN**

TEST: The amount of time before the timer expires is returned as the interval until timer expiration, even if the timer is armed with absolute time.

ELSE *NO_OPTION*

Conformance for timer_gettime: PASS, NO_OPTION

26 **IF** {_POSIX_REALTIME_SIGNALS}**THEN**

TEST: Only a single signal is queued to the process for a given timer at any point in time.

ELSE *NO_TEST_SUPPORT*

Conformance for timer_settime: PASS, NO_TEST_SUPPORT

27 **IF** {_POSIX_REALTIME_SIGNALS}**THEN**

TEST: When a timer for which a signal is still pending expires, and no signal is queued, a timer overrun occurs.

ELSE *NO_TEST_SUPPORT*

Conformance for timer_settime: PASS, NO_TEST_SUPPORT

timer_getoverrun

IF *PCTS_timer_getoverrun* **THEN**

IF {_POSIX_REALTIME_SIGNALS}**THEN**

TEST: When a timer expiration signal is delivered to a process, the *timer_getoverrun*() function returns the timer expiration overrun count for the specified timer.

NOTE: There is no known portable test method for this assertion.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for timer_getoverrun: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

28 **IF** *PCTS_timer_getoverrun* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: The PCD.1b documents the value of {DELAYTIMER_MAX} in 14.2.4.2.
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_OPTION

29 **IF** *PCTS_timer_getoverrun* **THEN**
TEST: The overrun count returned by *timer_getoverrun*() contains the number of extra timer expirations that occurred between the time the signal was generated (queued) and when it was delivered.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_TEST, NO_OPTION

30 **IF** *PCTS_timer_getoverrun* **THEN**
IF {DELAYTIMER_MAX} <= *PCTS_DELAYTIMER_MAX* **THEN**
TEST: When the number of extra expirations is greater than or equal to {DELAYTIMER_MAX}, then *timer_getoverrun*() returns {DELAYTIMER_MAX}.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

31 **IF** *PCTS_timer_getoverrun* **THEN**
TEST: The value returned by *timer_getoverrun*() applies to the most recent expiration signal delivery for the timer.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_TEST, NO_OPTION

D_2 **IF** *PCTS_timer_getoverrun* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the meaning of the overrun count returned, if no expiration signal has been delivered for the timer, or if {_POSIX_REALTIME_SIGNALS} is not supported, does so in 14.2.4.2.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_OPTION

D_3 **IF** *PCTS_timer_settime* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *timer_settime*() function does so in 14.2.4.2.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

D_4 **IF** *PCTS_timer_gettime* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *timer_gettime*() function does so in 14.2.4.2.
ELSE *NO_OPTION*
Conformance for timer_gettime: PASS, NO_OPTION

D_5 **IF** *PCTS_timer_getoverrun* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *timer_getoverrun*() function does so in 14.2.4.2.
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_OPTION

14.2.4.3 Returns

R_1 **IF** *PCTS_timer_settime* **THEN**

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: When a call to *timer_settime*() completes successfully, the interface returns a value of 0.

ELSE *NO_OPTION*

SEE: Assertion timer_settime in 14.2.4.4.

R_2 IF *PCTS_timer_gettime* **THEN**

TEST: When a call to *timer_gettime*() completes successfully, the interface returns a value of 0.

ELSE *NO_OPTION*

SEE: Assertion timer_gettime in 14.2.4.4.

R_3 IF *PCTS_timer_settime* **THEN**

TEST: When a call to *timer_settime*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.

timer_settime()

ELSE *NO_OPTION*

SEE: All assertions in 14.2.4.4 controlled by *timer_settime*().

R_4 IF *PCTS_timer_gettime* **THEN**

TEST: When a call to *timer_gettime*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.

timer_gettime()

ELSE *NO_OPTION*

SEE: All assertions in 14.2.4.4 controlled by *timer_gettime*().

R_5 IF *PCTS_timer_getoverrun* **THEN**

TEST: When a call to *timer_getoverrun*() completes successfully, the interface returns the timer expiration overrun count as explained in POSIX.1b {3}, 14.2.4.2.

ELSE *NO_OPTION*

SEE: Assertion timer_getoverrun in 14.2.4.4.

14.2.4.4 Errors

32 IF *PCTS_timer_settime* **THEN**

IF *PCTS_timer_create* and *PCTS_timer_delete* **THEN**

TEST: A call to *timer_settime*(), when the *timerid* argument does not correspond to an ID returned by *timer_create*() but not yet deleted by *timer_delete*(), returns a value of -1 and sets *errno* to [EINVAL].

NOTE: A subroutine is recommended that either returns an invalid timer ID, or indicates that there is no way to generate an invalid timer ID on the system.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

33 IF *PCTS_timer_gettime* **THEN**

IF *PCTS_timer_create* and *PCTS_timer_delete* **THEN**

TEST: A call to *timer_gettime*(), when the *timerid* argument does not correspond to an ID returned by *timer_create*() but not yet deleted by *timer_delete*(), returns a value of -1 and sets *errno* to [EINVAL].

NOTE: A subroutine is recommended that either returns an invalid timer ID, or indicates that there is no way to generate an invalid timer ID on the system.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for timer_gettime: PASS, NO_TEST_SUPPORT, NO_OPTION

34 IF *PCTS_timer_getoverrun* **THEN**

IF *PCTS_timer_create* and *PCTS_timer_delete* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: A call to *timer_getoverrun(, timerid)*, when the *timerid* argument does not correspond to an ID returned by *timer_create*() but not yet deleted by *timer_delete*(), returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid timer ID, or indicates that there is no way to generate an invalid timer ID on the system.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_TEST_SUPPORT, NO_OPTION

35 **IF** not *PCTS_timer_settime* **THEN**
TEST: A call to *timer_settime*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

36 **IF** not *PCTS_timer_gettime* **THEN**
TEST: A call to *timer_gettime*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for timer_gettime: PASS, NO_OPTION

37 **IF** not *PCTS_timer_getoverrun* **THEN**
TEST: A call to *timer_getoverrun* returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_OPTION

38 **IF** *PCTS_timer_settime* **THEN**
TEST: A call to *timer_settime(timerid, flags, value, ovalue*), when the *value* structure specifies a nanosecond value less than zero, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

39 **IF** *PCTS_timer_settime* **THEN**
IF {INT_MAX}<= 10e9 **THEN**
TEST: A call to *timer_settime(timerid, flags, value, ovalue*), when the *value* structure specifies a nanosecond greater than or equal to 1000 million, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

14.2.5 High Resolution Sleep

Function: *nanosleep*()

14.2.5.1 Synopsis

1
M_GA_stdC_proto_decl(int; nanosleep; const struct timespec *rmtp; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for nanosleep: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(nanosleep; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for nanosleep: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; nanosleep; time.h;;;)

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for nanosleep: PASS, NO_OPTION

4

M_GA_macro_args (nanosleep; time.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for nanosleep: PASS, NO_OPTION

14.2.5.2 Description

nanosleep

IF *PCTS_nanosleep* **THEN**
TEST: A successful call to *nanosleep(rqtp, rmtp)* causes the current process to be suspended from execution until the time interval specified by the *rqtp* argument has elapsed, whereupon it returns 0.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

5 **IF** *PCTS_nanosleep* **THEN**
TEST: In a successful call to *nanosleep(rqtp, rmtp)*, the suspension time is not less than the time specified by the *rqtp*, as measured by the system clock, CLOCK_REALTIME.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

6 **IF** *PCTS_nanosleep* **THEN**
TEST: In a successful call to *nanosleep(rqtp, rmtp)*, the suspension time may be longer than requested, because the argument value is rounded up to an integer multiple of the sleep resolution.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_TEST, NO_OPTION

7 **IF** *PCTS_nanosleep* **THEN**
TEST: In a successful call to *nanosleep(rqtp, rmtp)*, the suspension time may be longer than requested because of the scheduling of other activity by the system.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_TEST, NO_OPTION

8 **IF** *PCTS_nanosleep* **THEN**
TEST:
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

9 **IF** *PCTS_nanosleep* **THEN**
TEST: The use of the *nanosleep*() function has no effect on the action or blockage of any signal.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

D_1 **IF** *PCTS_nanosleep* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *nanosleep*() function does so in 14.2.5.2.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

14.2.5.3 Returns

R_1 **IF** *PCTS_nanosleep* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TEST: When a call to *nanosleep*() returns because the requested time has elapsed, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion nanosleep in 14.2.5.4.

R_2 IF *PCTS_nanosleep* **THEN**
TEST: When a call to *nanosleep*() returns because it has been interrupted by a signal, the interface returns a value of -1, and sets *errno* to indicate the interruption.
ELSE *NO_OPTION*
SEE: All assertions in 14.2.5.4.

10 IF *PCTS_nanosleep* **THEN**
TEST: When the *rmtp* argument is non-**NULL**, the *timespec* structure referenced by it is updated by *nanosleep(rqtp, rmtp)* to contain the amount of time remaining in the interval (the requested time minus the time actually slept).
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

11 IF *PCTS_nanosleep* **THEN**
TEST: When the *rmtp* argument is **NULL**, the remaining time is not returned by *nanosleep(rqtp, rmtp).*
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

R_3 IF *PCTS_nanosleep* **THEN**
TEST: When a call to *nanosleep*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 14.2.5.4.

14.2.5.4 Errors

12 IF *PCTS_nanosleep* **THEN**
TEST: A call to *nanosleep*(), when the *nanosleep*() function is interrupted by a signal, returns a value of -1 and sets *errno* to [EINTR].
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

13 IF *PCTS_nanosleep* **THEN**
TEST: A call to *nanosleep(rqtp, rmtp)*, when the *rqtp* argument specifies a nanosecond value less than zero, returns a value -1 and sets *errno* to [EINVAL].
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

14 IF *PCTS_nanosleep* **THEN**
IF INT_MAX<=10e9 **THEN**
TEST: A call to *nanosleep(rqtp, rmtp)*, when the *rqtp* argument specifies a nanosecond value greater than or equal to 1000 million, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_TEST_SUPPORT, NO_OPTION

15 IF not *PCTS_nanosleep* **THEN**
TEST: A call to *nanosleep*() returns a value of -1 and sets *errno* to [ENOSYS}.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Section 15: Message Passing

15.1 Data Definitions for Message Queues

D_1 SETUP: Include the header <mqueue.h>.

TEST: A PCD.1b that documents that the symbols allowed by this standard to be in the headers <sys/types.h>, <fcntl.h>, <time.h>, <signal.h>, are visible does so in 15.1.

ELSE *NO_OPTION*

Conformance for mq_intro: PASS, NO_OPTION

15.1.1 Data Structures

1 **SETUP:** Include the header <mqueue.h>.

TEST: The types *mqd_t* and *struct sigevent* are defined.

Conformance for mq_hdr: PASS

D_1 TEST: The PCD.1b documents the definition of *mqd_t* and *struct sigevent,* in 15.1.1.

Conformance for mq_hdr: PASS

3 **SETUP:** Include the header <mqueue.h>.

TEST: The structure *mq_attr* is defined and has at least the following members.

Member Type	Member Name	Description
long	*mq_flags*	Message queue flags
long	*mq_maxmsg*	Maximum number of messages
long	*mq_msgsize*	Maximum message size
long	*mq_curmsgs*	Number of messages currently queued

Conformance for mq_hdr: PASS

D_2 IF a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents extensions to *mq_attr*, as permitted in POSIX.1b{3}, 1.3.1.1, item (2), does so in 15.1.1.

ELSE *NO_OPTION*

Conformance for mq_hdr: PASS, NO_OPTION

4 **SETUP:** Include the header <mqueue.h>.

TEST: Extensions to *mq_attr* that may change the behavior of the application with respect to this standard when those fields in the structure are uninitialized, are enabled as required byPOSIX.1b {3} 1.3.1.1.

NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.

Conformance for mq_hdr: PASS, NO_TEST

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

D_3 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the existence of flags other than O_NONBLOCK, does so in 15.1.1.
ELSE *NO_OPTION*
Conformance for mq_hdr: PASS, NO_OPTION

M_GA_mqOpenMaxFD () =
IF *PCTS_MQ_FILE_DESCRIPTORS* **THEN**
IF ({OPEN_MAX} <= *PCTS_OPEN_MAX*) and *PCTS_mq_open* **THEN**
TEST: A process calling *mq_open*() can simultaneously open a combination of files and message queues totaling at least {OPEN_MAX}.
TR: Test for opening {OPEN_MAX} message queues.

Test for opening a message queue after opening {OPEN_MAX} -1 files.

Test for opening a file after opening {OPEN_MAX} -1 message queues.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*

GA_mqOpenMaxFD
FOR: *mq_open*()
M_GA_mqOpenMaxFD()
Conformance for mq_hdr: PASS, NO_OPTION

M_GA_mqPCTSOpenMaxFD () =
IF *PCTS_MQ_FILE_DESCRIPTORS* **THEN**
IF ({OPEN_MAX} > *PCTS_OPEN_MAX*) and *PCTS_mq_open* **THEN**
TEST: A process calling *mq_open*() can simultaneously open a combination of files and message queues totaling at least *PCTS_OPEN_MAX*.
TR: Test for opening *PCTS_OPEN_MAX* message queues.

Test for opening a message queue after opening *PCTS_OPEN_MAX*-1 files.

Test for opening a file after opening *PCTS_OPEN_MAX-1* message queues.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*

GA_mqPCTSOpenMaxFD
FOR: *mq_open*()
M_GA_mqPCTSOpenMaxFD()
Conformance for mq_hdr: PASS, NO_OPTION

15.2 Message Passing Functions

15.2.1 Open a Message Queue

Function: *mq_open*().

15.2.1.1 Synopsis

1

M_GA_stdC_proto_decl(mqd_t; mq_open; const char *name, int oflag, ... oflag; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for mq_open: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(mqd_t; mq_open; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for mq_open: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(mqd_t; mq_open; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for mq_open: PASS, NO_OPTION

4

M_GA_macro_args (mq_open; mqueue.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mq_open: PASS, NO_OPTION

15.2.1.2 Description

M_GA_mqOpenMaxFD()
SEE: Assertion GA_mqOpenMaxFD in 15.1.1.
Conformance for mq_open: PASS[OpenMaxMqs, PCTSOpenMaxMqs]

M_GA_mqPCTSOpenMaxFD()
SEE: Assertion GA_mqPCTSOpenMaxFD in 15.1.1.
Conformance for mq_open: PASS[OpenMaxMqs, PCTSOpenMaxMqs]

mq_open

IF *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A successful call to *mq_open*() establishes a connection between a process and a message queue, *name*, and returns a message queue, descriptor which can be used by other functions to refer to that message queue.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 IF *PCTS_mq_open* and a PCD.1b documents the following **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A PCD.1b that documents whether the name appears in the file system and is visible to other functions that take pathnames as arguments does so in 15.2.1.2.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

5

M_GA_portableFilenames(mq_open)
SEE: Assertion GA_portableFilenames in 2.2.2.40.
Conformance for mq_open: PASS, NO_OPTION

6

M_GA_upperLowerNames(mq_open)
SEE: Assertion GA_upperLowerNames in 2.2.2.40.
Conformance for mq_open: PASS, NO_OPTION

7

M_GA_PRNoTrunc(mq_open)
SEE: Assertion GA_PRNoTrunc in 2.3.6.
Conformance for mq_open: PASS, NO_OPTION

8

M_GA_PRNoTruncError(mq_open)
SEE: Assertion GA_PRNoTruncError in 2.2.2.40.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for mq_open: PASS, NO_OPTION

9 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When *name* begins with the slash character, then processes calling *mq_open(name, oflag, ...)* with the same value of *name* refer to the same message queue object, as long as that name has not been removed.
TR: Try the interface both in the same process that created the queue and in another process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_mq_open* **THEN**
TEST: The PCD.1b documents the effect of calling *mq_open(name, oflag, ...),* if *name* does not begin with the slash character, in 15.2.1.2.
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

D_3 **IF** *PCTS_mq_open* **THEN**
TEST: The PCD.1b documents the interpretation of slash characters, other than the leading slash character, in *name* in 15.2.1.2.
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

R_1 **IF** *PCTS_mq_open* **THEN**
TEST: When the *name* argument is not the name of an existing message queue and O_CREAT is not set, *mq_open(name, oflag, ...)* fails and returns an error.
ELSE *NO_OPTION*
SEE: Assertion mq_open_ENOENT in 15.2.1.4.

10 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: The access permission to receive messages or send messages, as specified by *oflag*, is granted if the calling process would be granted read or write access, respectively, to an equivalently protected file.

Read and write access to files is determined as described in 2.3.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **FOR:** mq_open
M_GA_AP_classAccess(mq_open)
SEE: Assertion GA_AP_class Access in 2.3.2.
Conformance for mq_open: PASS

12 **FOR:** mq_open
M_GA_AP_OVERRIDEFILEACCESS(mq_open)
SEE: Assertion GA_AP_overrideFile Access in 2.3.2.
Conformance for mq_open: PASS

13 **IF** *PCTS_mq_open* **THEN**
SETUP: Include the header `<mqueue.h>`.
TEST: The constants O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, and O_NONBLOCK are defined and have different values.
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

14 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_mq_send* and *PCTS_mq_receive* and *PCTS_GAP_mq_open* **THEN**
TEST: When a message queue is opened with the call *mq_open(name, oflag, ...)*, and the flag O_RDONLY is set in *oflag,* the process can use the returned message queue descriptor with *mq_receive*(), but not with *mq_send*().
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A message queue may be opened multiple times in the same or different processes with the call *mq_open(name, oflag, ...)* and the flag O_RDONLY set in *oflag.*
TR: Try the interface both in the same process that created the queue and in another process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_mq_open* and *PCTS_mq_send* and *PCTS_mq_receive* **THEN**
TEST: When a message queue is opened with the call *mq_open(name, oflag, ...)* and the flag O_WRONLY is set in *oflag,* the process can use the returned message queue descriptor with *mq_send*(), but not with *mq_receive*().
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

17 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A message queue may be opened multiple times in the same or different processes with the call *mq_open(name, oflag, ...)* and the flag O_WRONLY set in *oflag.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_mq_open* and *PCTS_mq_send* and *PCTS_mq_receive* **THEN**
TEST: When a message queue is opened with the call *mq_open(name, oflag, ...),* and the flag O_RDWR is set in *oflag,* the process can use any of the functions allowed for O_RDONLY and O_WRONLY.
TR: Try both *mq_send*() and *mq_receive*().
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When a message queue is opened with the call *mq_open(name, oflag, ...)*, and the flag O_RDWR is set in *oflag*, it may be open multiple times in the same or different processes for sending messages.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

20 **IF** *PCTS_mq_open* **THEN**

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IF *PCTS_GAP_mq_open* **THEN**
TEST: When a message queue is opened with the call *mq_open(name, oflag, mode, attr)* the named message queue does not already exist, and the flag O_CREAT is set in *oflag*, a message queue is created without any messages in it.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When *mq_open(name, oflag, mode, attr)* is called with O_CREAT set and O_EXCL cleared, and the named message queue already exists, the O_CREAT flag has no effect.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

22 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When a new message queue is created with *mq_open*(), the user ID of the message queue is set to the effective user ID of the process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

23 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When a new message queue is created with *mq_open*(), the group ID of the message queue is set to the effective group ID of the process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

24 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When a new message queue is created with *mq_open(name, oflag, ...)* , the "file permission bits" are set to the value of *mode.*
TR: Test for octal values 0000 through 0777.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

D_4 **IF** *PCTS_mq_open* **THEN**
TEST: The PCD.1b documents the effect when bits in *mode* are set, other than file permission bits, in 15.2.1.2.
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

D_5 **IF** *PCTS_mq_open* **THEN**
TEST: The PCD.1b documents the implementation-defined default message queue attributes with which message queues are created, if *mq_open(name, oflag, mode, attr)* is called and *attr* is **NULL**, in 15.2.1.2.
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

25 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When *attr* is non_**NULL** and the calling process has the appropriate privilege on *name*, the message queue *mq_maxmsg* and *mq_msgsize* attributes are set

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

to the values of the corresponding members in the *mq_attr* structure referred to by *attr*.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

R_2 IF *PCTS_mq_open* **THEN**

TEST: When *attr* is non_**NULL**, but the calling process does not have the appropriate privilege on *name*, the *mq_open*() function fails and returns an error without creating the message queue.

ELSE *NO_OPTION*

SEE: Assertion mq_open_EACCES2 in 15.2.1.4.

R_3 IF *PCTS_mq_open* **THEN**

TEST: When O_EXCL and O_CREAT are set, *mq_open*() fails if the message queue *name* exists.

ELSE *NO_OPTION*

SEE: Assertion mq_open_EEXIST in 15.2.1.4.

26 IF *PCTS_mq_open* **THEN**

IF *PCTS_GAP_mq_open* **THEN**

TEST: The check for the existence of the message queue and the creation of the message queue if it does not exist, is atomic with respect to other processes executing *mq_open*() naming the same *name* with O_EXCL and O_CREAT set.

There is no known reliable test method for this assertion.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

D_6 IF *PCTS_mq_open* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the result of calling *mq_open(name, oflag, ...),* if O_EXCL is set and O_CREAT is not set, does so in 15.2.2.2.

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_OPTION

27 IF *PCTS_mq_open* **THEN**

IF *PCTS_GAP_mq_open* **THEN**

TEST: The setting of O_NONBLOCK is associated with the open message queue descriptor, and determines whether a *mq_send*() or *mq_receive*() waits for resources or messages that are not currently available, or fails with *errno* set to [EAGAIN].

See *mq_send*() and *mq_receive*() for details.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

28 IF *PCTS_mq_open* **THEN**

IF *PCTS_GAP_mq_open* **THEN**

TEST: The *mq_open*() function does not add or remove messages from the queue.

TR: Open a message queue; send a single message; re-open the same queue; then try two successive reads.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

D_7 IF *PCTS_mq_open* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *mq_open*() function does so in 15.2.1.2.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

15.2.1.3 Returns

R_4 IF *PCTS_mq_open* **THEN**
TEST: When a call to *mq_open*() completes successfully, the function returns a message queue descriptor.
ELSE *NO_OPTION*
SEE: Assertion mq_open in 15.2.1.2.

R_5 IF *PCTS_mq_open* **THEN**
TEST: When a call to *mq_open*() completes unsuccessfully, the function returns *(mqd_t)* -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 15.2.1.4.

15.2.1.4 Errors

mq_open_EACCES1
IF *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open(name, oflag, ...)*, when the message queue exists and the permissions specified by *oflag* are denied, returns a value of -1 and sets *errno* to [EACCES].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

mq_open_EACCES2
IF *PCTS_mq_open* **THEN**
IF *PCTS_RAP_mq_open* **THEN**
TEST: A call to *mq_open(name, oflag, ...)*, when the message queue does not exist and permission to create the message queue is denied, returns a value of -1 and sets *errno* to [EACCES].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

mq_open_EEXIT
IF *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open(name, oflag, ...)*, when O_CREAT and O_EXCL are set and the named message queue already exists, returns a value of -1 and sets *errno* to [EEXIST].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

29 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open*(), when the *mq_open*() operation is interrupted by a signal, returns a value of -1 and sets *errno* to [EINTR].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

30 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open*(), when the *mq_open(name, oflag, ...)* operation is not supported for the given name , returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

D_8 **IF** *PCTS_mq_open* **THEN**
TEST: The PCD.1b documents under what circumstances the error [EINVAL] may be returned in 15.2.1.4.
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

31 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open(name, oflag, ...),* when O_CREAT is specified in *oflag*, the value of *attr* is not **NULL**, and *mq_maxmsg* is less than or equal to zero, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

32 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open(name, oflag, ...),* when O_CREAT is specified in *oflag*, the value of *attr* is not **NULL**, and *mq_msgsize* is less than or equal to zero, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

33 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* and ({OPEN_MAX} $\leq$ *PCTS_OPEN_MAX)* **THEN**
TEST: A call to *mq_open* (), when too many message queue descriptors or file descriptors are currently in use by this process, returns a value of -1 and sets *errno* to [EMFILE].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

34 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* and {MQ_OPEN_MAX} <= *PCTS_MQ_ OPEN_MAX)* **THEN**
TEST: A call to *mq_open* (), when too many message queue descriptors or file descriptors are currently in use by this process, returns a value of -1 and sets *errno* to [EMFILE].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

35 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* and {_POSIX_NO_TRUNC} and ({PATH_MAX} <= *PCTS_PATH_MAX)* **THEN**
TEST: A call to *mq_open (name, oflag, ...)*, when the length of the *name* string exceeds {PATH_MAX}, while {_POSIX_NO_TRUNC} is in effect, returns a value of -1 and sets *errno* to [ENAMETOOLONG].
ELSE *NO_TEST_SUPPORT*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

36 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* and {_POSIX_NO_TRUNC} and ({NAME_MAX} <= *PCTS_NAME_MAX)* **THEN**
TEST: A call to *mq_open (name, oflag, ...)*, when a pathname component is longer than {NAME_MAX}, while {_POSIX_NO_TRUNC} is in effect, returns a value of -1 and sets *errno* to [ENAMETOOLONG].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

37 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open* (), when too many message queues are currently open in the system, returns a value of -1 and sets *errno* to [ENFILE].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

mq_open_ENOENT
IF *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open* (), when O_CREAT is not set and the named message queue does not exist, returns a value of -1 and sets *errno* to [ENOENT].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

38 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open* (), when there is insufficient space for the creation of the new message queue, returns a value of -1 and sets *errno* to [ENOSPC].
There is no known reliable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

39 **IF** not *PCTS_mq_open* **THEN**
TEST: A call to *mq_open* () returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

15.2.2 Close a Message Queue

Function: *mq_close*().

15.2.2.1 Synopsis

1
M_GA_stdC_proto_decl(int; mq_close; mqd_t mqdes; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for mq_close: PASS[1, 2], NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2

M_GA_commonC_int_result_decl(mq_close; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for mq_close: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; mq_close; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for mq_close: PASS, NO_OPTION

4

M_GA_macro_args (mq_close; mqueue.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mq_close: PASS, NO_OPTION

15.2.2.2 Description

mq_close

IF *PCTS_mq_close* **THEN**
TEST: A call to *mq_close*() removes the association between the message queue descriptor, *mqdes*, and its message queue, and returns a value of zero.
ELSE *NO_OPTION*
Conformance for mq_close: PASS, NO_OPTION

D_1 **IF** *PCTS_mq_close* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the results of using this message queue descriptor after successful return from this *mq_close*(), and until the return of this message queue descriptor from a subsequent *mq_open*(), does so in 15.2.2.2.
ELSE *NO_OPTION*
Conformance for mq_close: PASS, NO_OPTION

5 **IF** *PCTS_mq_close* **THEN**
IF *PCTS_mq_notify* **THEN**
TEST: When the process has successfully attached a notification request to the message queue via *mqdes*, a call to *mq_close*() removes this attachment and makes the message queue available for another process to attach for notification.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_close: PASS,NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_mq_close* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mq_close*() function does so in 15.2.2.2.
ELSE *NO_OPTION*
Conformance for mq_close: PASS, NO_OPTION

15.2.2.3 Returns

R_1 **IF** *PCTS_mq_close* **THEN**
TEST: When a call to *mq_close*() completes successfully, it returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion mq_close in 15.2.2.2.

R_2 **IF** *PCTS_mq_close* **THEN**
TEST: When a call to *mq_close*() completes unsuccessfully, the function returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

SEE: All assertions in 15.2.2.4.

15.2.2.4 Errors

6 **IF** *PCTS_mq_close* **THEN**
TEST: A call to *mq_close(mqdes)*,when the *mqdes* argument is not a valid message queue descriptor, returns a value of -1 and sets *errno* to [EBADF].
ELSE *NO_OPTION*
Conformance for mq_close: PASS, NO_OPTION

7 **IF** not *PCTS_mq_close* **THEN**
TEST: A call to *mq_close*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mq_close: PASS, NO_OPTION

15.2.3 Remove a Message Queue

Function: *mq_unlink*().

15.2.3.1 Synopsis

1
M_GA_stdC_proto_decl(int; mq_unlink; const char *name; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for mq_unlink: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(mq_unlink; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for mq_unlink: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; mq_unlink; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for mq_unlink: PASS, NO_OPTION

4
M_GA_macro_args (mq_unlink; mqueue.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mq_unlink: PASS, NO_OPTION

15.2.3.2 Description

mq_unlink
IF *PCTS_mq_unlink* **THEN**
TEST: A successful call to *mq_link*() removes the message queue named by the pathname *name* and returns a value of 0.
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_OPTION

R_1 **IF** *PCTS_mq_unlink* **THEN**
IF *PCTS_mq_open* and *PCTS_gap_mq_open* **THEN**
TEST: After a successful call to *mq_unlink(name),* a call to *mq_open(name, oflag, ...)* fails if the flag O_CREAT is not set in *flags.*
ELSE *NO_TEST_SUPPORT*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
SEE: Assertion mq_unlink_ENOENT in 15.2.3.4.

5 **IF** *PCTS_mq_unlink* **THEN**
TEST: When one or more processes have the message queue open when *mq_unlink*() is called, destruction of the message queue is postponed until all references to the message queue have been closed.
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_OPTION

D_1 **IF** *PCTS_mq_unlink* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mq_link*() function does so in 15.2.3.4.
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_OPTION

15.2.3.3 Returns

R_2 **IF** *PCTS_mq_unlink* **THEN**
TEST: When a call to *mq_link*() completes successfully, the function returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion mq_unlink in 15.2.3.2.

R_3 **IF** *PCTS_mq_unlink* **THEN**
TEST: When a call to *mq_link*() completes unsuccessfully, the named message queue is unchanged and the function returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 15.2.3.4.

15.2.3.4 Errors

6 **IF** *PCTS_mq_unlink* **THEN**
TEST: A call to *mq_unlink(name),* when permission is denied to unlink the named message queue, returns a value of -1 and sets *errno* to [EACCES].
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_OPTION

7 **IF** *PCTS_mq_unlink* **THEN**
IF {_POSIX_NO_TRUNC} and ({NAME_MAX} <= *PCTS_NAME_MAX)* **THEN**
TEST: A call to *mq_unlink(name),* when the length of the *name* string exceeds {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect, returns a value of -1 and sets *errno* to [ENAMETOOLONG].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

mq_unlink_ENOENT
IF *PCTS_mq_unlink* **THEN**
TEST: A call to *mq_unlink(name),* when the named message queue does not exist, returns a value of -1 and sets *errno* to [ENOENT].
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_OPTION

8 **IF** not *PCTS_mq_unlink* **THEN**
TEST: A call to *mq_unlink*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for mq_unlink: PASS, NO_OPTION

15.2.4 Send a Message to a Message Queue

Function: *mq_send*().

15.2.4.1 Synopsis

1

M_GA_stdC_proto_decl(int; mq_send; mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for mq_send: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(mq_send; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for mq_send: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; mq_send; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for mq_send: PASS, NO_OPTION

4

M_GA_macro_args (mq_send; mqueue.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mq_send: PASS, NO_OPTION

15.2.4.2 Description

mq_send

IF *PCTS_mq_send* **THEN**
TEST: A successful call to *mq_send*() adds the message pointed to by the argument *msg_ptr* to the message queue specified by *mqdes*, and returns a value of zero.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

R_1 **IF** *PCTS_mq_send* **THEN**
TEST: When the value of *msg_len*, which specifies the length of the message in bytes pointed to by *msg_ptr,* is greater than the *mq_msgsize* attribute of the message queue, *mq_send*() fails.
ELSE *NO_OPTION*
SEE: Assertion mq_send_EMSGSIZE in 15.2.4.4.

5 **IF** *PCTS_mq_send* **THEN**
TEST: When the specified message queue is not full, *mq_send*() behaves as if the message is inserted into the message queue at the position indicated by the *msg_prio* argument.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

6 **IF** *PCTS_mq_send* **THEN**
TEST: A message with a larger numeric value of *msg_prio* is inserted before messages with lower values of *msg_prio.*
ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for mq_send: PASS, NO_OPTION

7 **IF** *PCTS_mq_send* **THEN**
TEST: A message is inserted after other messages in the queue, if any, with equal *msg_prio*.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

R_2 **IF** *PCTS_mq_send* **THEN**
TEST: The value of *msg_prio* is less than or equal to {MQ_PRIO_MAX}.
ELSE *NO_OPTION*
SEE: Assertion mq_send_EINVAL in 15.2.4.4.

8 **IF** *PCTS_mq_send* **THEN**
TEST: When the specified message queue is full and O_NONBLOCK is not set in the message queue description associated with *mqdes*, *mq_send*() blocks until space becomes available to enqueue the message, or until *mq_send*() is interrupted by a signal.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

9 **IF** *PCTS_mq_send* **THEN**
IF {_POSIX_PRIORITY_SCHEDULING} **THEN**
TEST: When more than one process is waiting to send when space becomes available in the message queue, and the {_POSIX_PRIORITY_SCHEDULING} option is supported, then the process of the highest priority that has been waiting the longest is unblocked to send its message.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_mq_send* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents which waiting process is unblocked does so in 15.2.4.2.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

R_3 **IF** *PCTS_mq_send* **THEN**
TEST: When the specified message queue is full, and O_NONBLOCK is set in the message queue description associated with *mqdes*, the message is not queued and *mq_send*() returns an error.
ELSE *NO_OPTION*
SEE: Assertion mq_send_EAGAIN in 15.2.4.4.

D_2 **IF** *PCTS_mq_send* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mq_send*() function does so in 15.2.4.2.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

15.2.4.3 Returns

R_4 **IF** *PCTS_mq_send* **THEN**
TEST: When a call to *mq_send*() completes successfully, it returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion mq_send in 15.2.4.2.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

R_5 IF *PCTS_mq_send* **THEN**
TEST: When a call to *mq_send*() completes unsuccessfully, no message is enqueued, the function returns -1 and *errno* is set to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 15.2.4.4.

15.2.4.4 Errors

mq_send_EAGAIN
IF *PCTS_mq_send* **THEN**
TEST: A call to *mq_send(mqdes, msg_ptr, msg_len, msg_prio),* when the O_NONBLOCK flag is set in the message queue description associated with *mqdes*, and the specified message queue is full, returns a value of -1 and sets *errno* to [EAGAIN].
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

10 **IF** *PCTS_mq_send* **THEN**
TEST: A call to *mq_send(mqdes, msg_ptr, msg_len, msg_prio),* when the *mqdes* argument is not a valid message queue descriptor open for writing, returns a value of -1 and sets *errno* to [EBADF].
TR: Open a message queue O_RDONLY; then send to that queue.

Send to an invalid message queue.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

11 **IF** *PCTS_mq_send* **THEN**
TEST: A call to *mq_send*(), when a signal interrupts the call, returns a value of -1 and sets *errno* to [EINTR].
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

mq_send_EINVAL
IF *PCTS_mq_send* **THEN**
IF {MQ_PRIO_MAX} < {UINT_MAX} **THEN**
TEST: A call to *mq_send(mqdes, msg_ptr, msg_len, msg_prio),* when the value of *msg_prio* is outside the valid range, returns a value of -1 and sets *errno* to [EINVAL].
TR: Try a value of {MQ_PRIO_MAX}+1 if it is less than {UINT_MAX}.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_TEST_SUPPORT, NO_OPTION

mq_send_EMSGSIZE
IF *PCTS_mq_send* **THEN**
TEST: A call to *mq_send(mqdes, msg_ptr, msg_len, msg_prio),* when the specified message length *msg_len* exceeds the message size attribute of the message queue, returns a value of -1 and sets *errno* to [EMSGSIZE].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

12 **IF** not *PCTS_mq_send* **THEN**
TEST: A call to *mq_send*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15.2.5 Receive a Message From a Message Queue

Function: *mq_receive*().

15.2.5.1 Synopsis

1

M_GA_stdC_proto_decl(ssize_t; mq_receive; mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for mq_receive: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(ssize_t; mq_receive; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for mq_receive: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(ssize_t; mq_receive; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for mq_receive: PASS, NO_OPTION

4

M_GA_macro_args (mq_receive; mqueue.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mq_receive: PASS, NO_OPTION

15.2.5.2 Description

mq_receive

IF *PCTS_mq_receive* **THEN**
TEST: A successful call to *mq_receive* () receives the oldest of the highest priority message(s) from the message queue specified by *mqdes*, copying it to the buffer pointed to by the *msg_ptr* argument, removing it from the queue, and returning its length in bytes.
ELSE *NO_OPTION*
Conformance for mq_receive: PASS, NO_OPTION

5 **IF** *PCTS_mq_receive* **THEN**
TEST: When the size of the buffer in bytes, specified by the *msg_len* argument, is less than the *mq_msgsize* attribute of the message queue, the function *mq_receive(mqdes, msg_ptr, msg_len, msg_prio)* fails and returns an error.
ELSE *NO_OPTION*
Conformance for mq_receive: PASS, NO_OPTION

6 **IF** *PCTS_mq_receive* **THEN**
TEST: When the argument *msg_prio* is not **NULL**, the priority of the selected message is stored in the location referenced by *msg_prio*.
ELSE *NO_OPTION*
Conformance for mq_receive: PASS, NO_OPTION

7 **IF** *PCTS_mq_receive* **THEN**
TEST: When the specified message queue is empty, and O_NONBLOCK is not set in the message queue description associated with *mqdes*, *mq_receive*() blocks until a

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

message is enqueued on the message queue, or until *mq_receive*() is interrupted by a signal.

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_OPTION

8 **IF** *PCTS_mq_receive* **THEN**

IF {_POSIX_PRIORITY_SCHEDULING} **THEN**

TEST: When more than one process is waiting to receive a message when a message arrives at an empty queue, and the {_POSIX_PRIORITY_SCHEDULING} option is supported, then the process of highest priority that has been waiting the longest is selected to receive the message.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_mq_receive* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents which waiting process receives the message, if more than one process is waiting to receive a message when a message arrives at an empty queue, and the {_POSIX_PRIORITY_SCHEDULING} option is not supported, does so in 15.2.5.2.

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_OPTION

R_1 **IF** *PCTS_mq_receive* **THEN**

TEST: When the specified message queue is empty, and O_NONBLOCK is set in the message queue description associated with *mqdes*, no message is removed from the queue and *mq_receive*() returns an error.

ELSE *NO_OPTION*

SEE: Assertion for mq_receive_EAGAIN in 15.2.5.4.

D_2 **IF** *PCTS_mq_receive* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *mq_receive*() function does so in 15.2.5.2.

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_OPTION

15.2.5.3 Returns

R_2 **IF** *PCTS_mq_receive* **THEN**

TEST: When a call to *mq_receive*() completes successfully, it returns the length of the selected message in bytes and the message is removed from the queue.

ELSE *NO_OPTION*

SEE: Assertion for mq_receive in 15.2.5.2.

R_3 **IF** *PCTS_mq_receive* **THEN**

TEST: When a call to *mq_receive*() completes unsuccessfully, no message is removed from the queue, the function returns a value of -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: All assertions in 15.2.5.4.

15.2.5.4 Errors

mq_receive_EAGAIN

IF *PCTS_mq_receive* **THEN**

TEST: A test to *mq_receive(mqdes, msg_ptr, msg_len, msg_prio)*, when O_NONBLOCK is set in the message description associated with *mqdes*, and the specified message queue is empty, returns a value of -1 and sets *errno* to [EAGAIN].

ELSE *NO_OPTION*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for mq_receive: PASS, NO_OPTION

9 **IF** *PCTS_mq_receive* **THEN**

TEST: A call to *mq_receive(mqdes, msg_ptr, msg_len, msg_prio)*, when the *mqdes* argument is not a valid message queue descriptor open for reading, returns a value of -1 and sets *errno* to [EBADF].

TR: Open a message queue O_WRONLY; then receive from that queue.

Receive from an invalid message queue.

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_OPTION

10 **IF** *PCTS_mq_receive* **THEN**

TEST: A call to *mq_receive(mqdes, msg_ptr, msg_len, msg_prio)*, when the specified message buffer size, *msg_len*, is less than the message size attribute of the message queue, returns a value of -1 and sets *errno* to [EMSGSIZE].

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_OPTION

11 **IF** *PCTS_mq_receive* **THEN**

TEST: A call to *mq_receive*(), when operation is interrupted by a signal, returns a value of -1 and sets *errno* to [EINTR].

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_OPTION

12 **IF** not *PCTS_mq_receive* **THEN**

TEST: A call to *mq_receive*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_OPTION

13 **IF** *PCTS_mq_receive* **THEN**

IF *PCTS_DETECT_MESSAGE_DATA_CORRUPTION* **THEN**

TEST: A call to *mq_receive*(), when the implementation has detected a data corruption problem with the message, returns a value of -1 and sets *errno* to [EBADMSG].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_TEST_SUPPORT, NO_OPTION

15.2.6 Notify Process that a Message is Available on a Queue

Function: *mq_notify*().

15.2.6.1 Synopsis

1

M_GA_stdC_proto_decl(int; mq_notify; mqd_t mqdes, const struct sigevent *notification; mqueue.h;;;)

SEE: Assertion GA_stdC_proto_decl in 2.7.3.

Conformance for mq_notify: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(mq_notify; mqueue.h;;;)

SEE: Assertion GA_commonC_int_result_decl in 2.7.3.

Conformance for mq_notify: PASS[1, 2], NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3

M_GA_macro_result_decl(int; mq_notify; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for mq_notify: PASS, NO_OPTION

4

M_GA_macro_args (mq_notify; mqueue.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mq_notify: PASS, NO_OPTION

15.2.6.2 Description

mq_notify

IF *PCTS_mq_notify* **THEN**
TEST: A call to *mq_notify(mqdes, notification)*, if the argument *notification* is not **NULL**, registers the calling process to be notified of message arrival at an empty message queue associated with the specified message queue descriptor *mqdes* and returns a value of 0.

The notification specified by the *notification* argument is sent to the process when the message queue transitions from empty to nonempty.
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

R_1 **IF** *PCTS_mq_notify* **THEN**
TEST: At any time, only one process may be registered for notification by a message queue. When the calling process, or any other process, has already registered for notification of message arrival at the specified message queue, subsequent attempts to register for that message queue fail.
ELSE *NO_OPTION*
SEE: Assertion mq_notify_EBUSY in 15.2.5.4.

5 **IF** *PCTS_mq_notify* **THEN**
TEST: When *notification* is **NULL**, and the process is currently registered for notification by the specified message queue, the existing registration for notification is removed.
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

6 **IF** *PCTS_mq_notify* **THEN**
TEST: When notification is sent to the registered process, its registration is removed and the message queue is made available for registration.
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

7 **IF** *PCTS_mq_notify* **THEN**
IF *PCTS_mq_receive* **THEN**
TEST: When a process has registered for notification of message arrival at a message queue, and some process is blocked in *mq_receive*() waiting to receive a message when a message arrives at the queue, the arriving message satisfies the appropriate *mq_receive*() (see POSIX.1b {3}, 15.2.5).

The resulting behavior is as if the message queue remains empty, and no notification is sent.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_mq_notify* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mq_notify*() function does so in 15.2.6.2.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

15.2.6.3 Returns

R_2 **IF** *PCTS_mq_notify* **THEN**
TEST: When a call to *mq_notify*() completes successfully, it returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion mq_notify in 15.2.6.2.

R_3 **IF** *PCTS_mq_notify* **THEN**
TEST: When a call to *mq_notify*() completes unsuccessfully, it returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 15.2.6.4.

15.2.6.4 Errors

8 **IF** *PCTS_mq_notify* **THEN**
TEST: A call to *mq_notify(mqdes, notification)*, when the *mqdes* argument is not a valid message queue descriptor, returns a value of -1 and sets *errno* to [EBADF].
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

mq_notify_EBUSY
IF *PCTS_mq_notify* **THEN**
TEST: A call to *mq_notify*(), when a process is already registered for notification by the message queue, returns a value of -1 and sets *errno* to [EBUSY].
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

9 **IF** not *PCTS_mq_notify* **THEN**
TEST: A call to *mq_notify*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

15.2.7 Set Message Queue Attributes

Function: *mq_setattr*().

15.2.7.1 Synopsis

1
M_GA_stdC_proto_decl(int; mq_setattr; mqd_t mqdes, const struct mq_attr *mqstat, struct mq_attr *omqstat; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for mq_setattr: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(mq_setattr; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for mq_setattr: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; mq_setattr; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Conformance for mq_setattr: PASS, NO_OPTION

4

M_GA_macro_args (mq_setattr; mqueue.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mq_setattr: PASS, NO_OPTION

15.2.7.2 Description

mq_setattr

IF *PCTS_mq_setattr* **THEN**
TEST: A call to *mq_setattr*() sets attributes associated with the message queue specified by *mqdes* and returns a value of zero.
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

5 **IF** *PCTS_mq_setattr* **THEN**
TEST: The message queue attributes corresponding to *mq_flags*, defined in the *mq_attr* structure, are set to the specified values upon successful completion of *mq_setattr*().
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

D_1 **IF** *PCTS_mq_setattr* **THEN**
TEST: The PCD.1b documents any implementation-defined flags that can be set in *mq_flags* in 15.2.7.2.
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

6 **IF** *PCTS_mq_setattr* **THEN**
TEST: The values of the *mq_maxmsg, mq_msgsize,* and *mg_curmsgs* members of the *mq_attr* structure are ignored by *mq_setattr*().
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

7 **IF** *PCTS_mq_setattr* **THEN**
IF *PCTS_mq_getattr* **THEN**
TEST: When *omqstat* is non-**NULL**, the function *mq_setattr*() stores, in the location referenced by *omqstat*, the previous message queue attributes and the current queue status.

These values are the same as would be returned by a call to *mq_getattr*() at that point.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_mq_setattr* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mq_setattr*() function does so in 15.2.7.2.
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

15.2.7.3 Returns

R_1 **IF** *PCTS_mq_setattr* **THEN**
TEST: When a call to *mq_setattr*() completes successfully, the function returns a value of 0 and changes the attributes of the message queue as specified.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ELSE *NO_OPTION*
SEE: Assertion mq_setattr in 15.2.7.2.

R_2 IF *PCTS_mq_setattr* **THEN**
TEST: When a call to *mq_setattr*() completes unsuccessfully, the message queue attributes are unchanged, and the function returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in 15.2.7.4.

15.2.7.4 Errors

9 **IF** not *PCTS_mq_setattr* **THEN**
TEST: A call to *mq_setattr*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

15.2.8 Get Message Queue Attributes

Function: *mq_getattr*()

15.2.8.1 Synopsis

1
M_GA_stdC_proto_decl(int; mq_getattr; mqd_t mqdes, struct mq_attr *mqstat; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in 2.7.3.
Conformance for mq_getattr: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(mq_getattr; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in 2.7.3.
Conformance for mq_getattr: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; mq_getattr; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in 1.3.4.
Conformance for mq_getattr: PASS, NO_OPTION

4
M_GA_macro_args (mq_getattr; mqueue.h;;;)
SEE: Assertion GA_macro_args in 2.7.3.
Conformance for mq_getattr: PASS, NO_OPTION

15.2.8.2 Description

mq_getattr
IF *PCTS_mq_getattr* **THEN**
TEST: A call to *mq_getattr*() gets status information and attributes associated with the message queue specified in *mqdes* and returns 0.
ELSE *NO_OPTION*
Conformance for mq_getattr: PASS, NO_OPTION

5 **IF** *PCTS_mq_getattr* **THEN**
IF *PCTS_mq_setattr* **THEN**
TEST: After a successful call to *mq_getattr(mqdes, mqstat)*, the *mq_flags* member within the *mq_attr* structure referenced by the *mqstat* argument has the value that was set

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

when the message queue was created, but with modifications made by subsequent *mq_setattr*() calls.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_getattr: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** *PCTS_mq_getattr* **THEN**

TEST: After a successful call to *mq_getattr(mqdes, mqstat)*, the *mq_maxmsg* and the *mq_msgsize* members within the *mq_attr* structure referenced by the *mqstat* argument have the values that were set at message queue creation.

ELSE *NO_OPTION*

Conformance for mq_getattr: PASS, NO_OPTION

7 **IF** *PCTS_mq_getattr* **THEN**

TEST: After a successful call to *mq_getattr(mqdes, mqstat)*, the *mq_curmsgs* member within the *mq_attr* structure referenced by the *mqstat* argument has the value that is set according to the number of messages currently on the queue.

ELSE *NO_OPTION*

Conformance for mq_getattr: PASS, NO_OPTION

D_1 **IF** *PCTS_mq_getattr* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *mq_getattr*() function does so in 15.2.8.2.

ELSE *NO_OPTION*

Conformance for mq_getattr: PASS, NO_OPTION

15.2.8.3 Returns

R_1 **IF** *PCTS_mq_getattr* **THEN**

TEST: When a call to *mq_getattr*() completes successfully, it returns 0.

ELSE *NO_OPTION*

SEE: Assertion mq_getattr in 15.2.8.2.

R_2 **IF** *PCTS_mq_getattr* **THEN**

TEST: When a call to *mq_getattr*() completes unsuccessfully, the function returns -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: All assertions in 15.2.8.4.

8 **IF** *PCTS_mq_getattr* **THEN**

TEST:

ELSE *NO_OPTION*

Conformance for mq_getattr: PASS, NO_OPTION

15.2.8.4 Errors

9 **IF** *PCTS_mq_getattr* **THEN**

TEST: A call to *mq_getattr(mqdes, mqstat)*, when the *mqdes* argument is not a valid message queue descriptor, returns a value of -1 and sets *errno* to [EBADF].

ELSE *NO_OPTION*

Conformance for mq_getattr: PASS, NO_OPTION

10 **IF** not *PCTS_mq_getattr* **THEN**

TEST: A call to *mq_getattr*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for mq_getattr: PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex A
(normative)

Conforming Test Results

A.1 General

Assertions for conformance

Clause	Assertion ID	Conforming Results
1.3.3.3	GD_CommonC_diffs	General Documentation Assertion, no test results
1.3.4	GA_macro_args	General Assertion, no test results
1.3.4	GA_macro_result_decl	General Assertion, no test results

A.2 Terminology and General Requirements

Assertions for definitions

Clause	Assertion ID	Conforming Results
2.2.2.40	GA_portableFilenames	General Assertion, no test results
2.2.2.40	GA_upperLowerNames	General Assertion, no test results
2.2.2.105	D_1	PASS
2.2.2.105	D_2	PASS
2.2.2.108	R_1	Reference Assertion, no test results
2.2.2.109	R_2	Reference Assertion, no test results
2.2.2.119	GA_syncIODataIntegrityRead	General Assertion, no test results
2.2.2.119	GA_syncIODataIntegrityWbeforeR	General Assertion, no test results
2.2.2.119	GA_syncIODataIntegrityWrite	General Assertion, no test results
2.2.2.120	GA_syncIOFileIntegrityRead	General Assertion, no test results
2.2.2.120	GA_syncIOFileIntegrityWrite	General Assertion, no test results
2.2.2.126	D_3	PASS

Assertions for General Concepts

Clause	Assertion ID	Conforming Results
2.3.2	GA_AP_overrideFileAccess	General Assertion, no test results
2.3.2	GA_AP_overrideExecAccess	General Assertion, no test results
2.3.2	GA_AP_classAccess	General Assertion, no test results

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.3.2	GA_AdditionalAccessControl	General Assertion, no test results
2.3.2	GA_AlternateAccessControl	General Assertion, no test results
2.3.2	GA_AltAccessEnable	General Assertion, no test results
2.3.2	GA_AltAccessDisable	General Assertion, no test results
2.3.5	GA_StatTimeUpdate	General Assertion, no test results
2.3.5	GA_NoOpenTimeUpdate	General Assertion, no test results
2.3.5	GA_NoROFSTimeUPDATE	General Assertion, no test results
2.3.6	GA_PRDot	General Assertion, no test results
2.3.6	GA_PRSlash	General Assertion, no test results
2.3.6	GA_PR3SLASH	General Assertion, no test results
2.3.6	GA_PRDotDot	General Assertion, no test results
2.3.6	GA_PRRelativeSlash	General Assertion, no test results
2.3.6	GA_PRRelativeSlashSlash	General Assertion, no test results
2.3.6	GA_PRRenameRelativeSlashSlash	General Assertion, no test results
2.3.6	GA_PRRelativeCWD	General Assertion, no test results
2.3.6	GA_PRRelativeDotCWD	General Assertion, no test results
2.3.6	GA_PRRelativeDotDotCWD	General Assertion, no test results
2.3.6	GA_PRRelativeSlashSlashCWD	General Assertion, no test results
2.3.6	GA_PRNoTrunc	General Assertion, no test results
2.3.6	GA_PRNoTruncError	General Assertion, no test results

Assertions for Error Numbers

Clause	Assertion ID	Conforming Results
2.4	2	PASS
2.4	GD_OptionalErrors	General Documentation Assertion, no test results
2.4	GA_Optional ErrorsUndetected	General Assertion, no test results

Assertions for Environment Description

Clause	Assertion ID	Conforming Results
2.6	GA_ExecNoSlash	General Assertion, no test results
2.6	GA_ExecColon	General Assertion, no test results
2.6	GA_ExecInsertSlash	General Assertion, no test results
2.6	GA_ExecTwoColons	General Assertion, no test results
2.6	GA_ExecInitialColon	General Assertion, no test results
2.6	GA_ExecTrailingColon	General Assertion, no test results
2.6	GA_ExecPathSearchOrder	General Assertion, no test results
2.6	GA_EnvironCaseSensitive	General Assertion, no test results
2.6	GA_EnvironPortNames	General Assertion, no test results

Assertions for C Language Definitions

Clause	Assertion ID	Conforming Results
2.7.1	4	PASS
2.7.1	4.1	PASS
2.7.2	5	PASS, NO_OPTION, NO_TEST
2.7.2	6	PASS, NO_OPTION, NO_TEST

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.7.2	D_1	PASS, NO_OPTION
2.7.2.1	7	PASS, NO_TEST
2.7.2.1	8	PASS, NO_TEST
2.7.2.2	9	PASS, NO_TEST
2.7.3	GAstdC_proto_decl	General Assertion, no test results
2.7.3	GA_commonC_result_decl	General Assertion, no test results
2.7.3	GA_commonC_int_result_decl	General Assertion, no test results
2.7.3	GA_setjmpDecl	General Assertion, no test results
2.7.3	GA_sigsetjmpDecl	General Assertion, no test results
2.7.3	GA_macro_args	General Assertion, no test results
2.7.3	D_2	PASS, NO_OPTION

Assertions for Numerical Limits

Clause	Assertion ID	Conforming Results
2.8.2	2	PASS
2.8.4	4	PASS
2.8.4	5	PASS, NO_OPTION
2.8.4	D_2	PASS
2.8.7	18	PASS

Assertions for Symbolic Constants

Clause	Assertion ID	Conforming Results
2.9	D_1	PASS
2.9	D_2	PASS
2.9.3	1	PASS
2.9.3	2	PASS
2.9.3	3	PASS, NO_OPTION
2.9.3	4	PASS, NO_OPTION
2.9.3	5	PASS, NO_OPTION
2.9.4	6	PASS
2.9.4	7	PASS, NO_TEST
2.9.4	8	PASS, NO_TEST

A.3 Process Primitives

Assertions for fork

Clause	Assertion ID	Conforming Results
3.1.1.2	1	PASS, NO_OPTION
3.1.1.2	2	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.1.2	3	PASS, NO_OPTION
3.1.1.2	4	PASS, NO_OPTION
3.1.1.2	5	PASS, NO_OPTION
3.1.1.2	6	PASS, NO_OPTION
3.1.1.2	7	PASS, NO_OPTION
3.1.1.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
3.1.1.2	9	PASS, NO_OPTION
3.1.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.1.1.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.1.2	12	PASS, NO_OPTION, NO_TEST
3.1.1.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for execl

Clause	Assertion ID	Conforming Results
3.1.2.2	1	PASS, NO_OPTION
3.1.2.2	2	PASS, NO_OPTION
3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	D_1	PASS, NO_OPTION
3.1.2.2	4	PASS, NO_OPTION, NO_TEST
3.1.2.2	5	PASS, NO_OPTION
3.1.2.2	D_2	PASS, NO_OPTION
3.1.2.2	6	PASS, NO_OPTION, NO_TEST
3.1.2.2	7	PASS, NO_OPTION
3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	9	PASS, NO_OPTION, NO_TEST SUPPORT
3.1.2.2	10	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_3	PASS, NO_OPTION
3.1.2.2	11	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_4	PASS, NO_OPTION

Assertions for execv

Clause	Assertion ID	Conforming Results
3.1.2.2	1	PASS, NO_OPTION
3.1.2.2	2	PASS, NO_OPTION
3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	D_1	PASS, NO_OPTION
3.1.2.2	4	PASS, NO_OPTION, NO_TEST
3.1.2.2	5	PASS, NO_OPTION
3.1.2.2	D_2	PASS, NO_OPTION
3.1.2.2	6	PASS, NO_OPTION, NO_TEST
3.1.2.2	7	PASS, NO_OPTION
3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	10	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_3	PASS, NO_OPTION
3.1.2.2	11	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_4	PASS, NO_OPTION

Assertions for execle

Clause	Assertion ID	Conforming Results
3.1.2.2	1	PASS, NO_OPTION
3.1.2.2	2	PASS, NO_OPTION
3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	D_1	PASS, NO_OPTION
3.1.2.2	4	PASS, NO_OPTION, NO_TEST

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.1.2.2	5	PASS, NO_OPTION
3.1.2.2	D_2	PASS, NO_OPTION
3.1.2.2	6	PASS, NO_OPTION, NO_TEST
3.1.2.2	7	PASS, NO_OPTION
3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	10	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_3	PASS, NO_OPTION
3.1.2.2	11	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_4	PASS, NO_OPTION

Assertions for execlp

Clause	Assertion ID	Conforming Results
3.1.2.2	1	PASS, NO_OPTION
3.1.2.2	2	PASS, NO_OPTION
3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	D_1	PASS, NO_OPTION
3.1.2.2	4	PASS, NO_OPTION, NO_TEST
3.1.2.2	5	PASS, NO_OPTION
3.1.2.2	D_2	PASS, NO_OPTION
3.1.2.2	6	PASS, NO_OPTION, NO_TEST
3.1.2.2	7	PASS, NO_OPTION
3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	10	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_3	PASS, NO_OPTION
3.1.2.2	11	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_4	PASS, NO_OPTION

Assertions for execvp

Clause	Assertion ID	Conforming Results
3.1.2.2	1	PASS, NO_OPTION
3.1.2.2	2	PASS, NO_OPTION
3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	D_1	PASS, NO_OPTION
3.1.2.2	4	PASS, NO_OPTION, NO_TEST
3.1.2.2	5	PASS, NO_OPTION
3.1.2.2	D_2	PASS, NO_OPTION
3.1.2.2	6	PASS, NO_OPTION, NO_TEST
3.1.2.2	7	PASS, NO_OPTION
3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	10	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_3	PASS, NO_OPTION
3.1.2.2	11	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_4	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for Process Termination

Clause	Assertion ID	Conforming Results
No assertions for Process Termination		

Assertions for wait

Clause	Assertion ID	Conforming Results
No assertions for wait		

Assertions for _exit

Clause	Assertion ID	Conforming Results
3.2.2.2	1	PASS, NO_OPTION, NO_TEST_SUPPORT
3.2.2.2	2	PASS, NO_OPTION, NO_TEST_SUPPORT
3.2.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
3.2.2.2	4	PASS, NO_OPTION, NO_TEST
3.2.2.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
3.2.2.2	6	PASS, NO_OPTION, NO_TEST
3.2.2.2	D_1	PASS, NO_OPTION
3.2.2.2	D_2	PASS

Assertions for signal.h

Clause	Assertion ID	Conforming Results
3.3.1.1	1	PASS
3.3.1.1	2	PASS, NO_OPTION
3.3.1.1	3	PASS
3.3.1.1	4	PASS
3.3.1.1	5	PASS
3.3.1.1	D_1	PASS
3.3.1.2	6	PASS
3.3.1.2	D_2	PASS, NO_OPTION
3.3.1.2	7	PASS
3.3.1.2	D_3	PASS, NO_OPTION
3.3.1.2	GA_sigev_value	General Assertion, no test results
3.3.1.2	GA_sigqueueValue	General Assertion, no test results
3.3.1.2	GA_sigPending Queued	General Assertion, no test results
3.3.1.2	D_4	PASS, NO_OPTION
3.3.1.2	ga_queuedAndRegularSignals	General Assertion, no test results
3.3.1.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.3	12	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.3	13	PASS, NO_OPTION
3.3.1.3	14	PASS, NO_OPTION, NO_TEST
3.3.1.3	D_5	PASS, NO_OPTION
3.3.1.3	15	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.3	16	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
3.3.1.3	17	PASS, NO_OPTION, NO_TEST_SUPPORT`
3.3.1.3	18	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.3.1.3	19	PASS, NO_OPTION
3.3.1.3	20	PASS
3.3.1.3	21	PASS, NO_OPTION
3.3.1.3	22	PASS
3.3.1.3	D_6	PASS, NO_OPTION
3.3.1.3	23	PASS, NO_OPTION
3.3.1.3	24	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.3	25	PASS, NO_OPTION
3.3.1.3	26	PASS, NO_OPTION
3.3.1.3	D_7	PASS
3.3.1.3	27	PASS, NO_OPTION
3.3.1.3	28	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.3	29	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.3	30	PASS, NO_OPTION
3.3.1.3	31	PASS, NO_OPTION
3.3.1.3	D_8	PASS, NO_OPTION
3.3.1.3	D_9	PASS, NO_OPTION
3.3.1.3	32	PASS, NO_OPTION
3.3.1.3	33	PASS, NO_OPTION
3.3.1.3	34	PASS, NO_OPTION
3.3.1.3	35	PASS, NO_OPTION
3.3.1.3	36	PASS, NO_OPTION
3.3.1.3	37	PASS, NO_OPTION
3.3.1.3	38	PASS, NO_OPTION
3.3.1.3	39	PASS, NO_OPTION
3.3.1.3	40	PASS, NO_OPTION
3.3.1.3	41	PASS, NO_OPTION

Assertions for kill

Clause	Assertion ID	Conforming Results
No assertions for kill		

Assertions for sigemptyset

Clause	Assertion ID	Conforming Results
No assertions for sigemptyset		

Assertions for sigfillset

Clause	Assertion ID	Conforming Results
No assertions for sigfillset		

Assertions for sigaddset

Clause	Assertion ID	Conforming Results
No assertions for sigaddset		

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for sigdelset

Clause	Assertion ID	Conforming Results
No assertions for sigdelset		

Assertions for sigismember

Clause	Assertion ID	Conforming Results
No assertions for sigismember		

Assertions for sigaction

Clause	Assertion ID	Conforming Results
3.3.4.2	1	PASS
3.3.4.2	2	PASS, NO_OPTION
3.3.4.2	3	PASS
3.3.4.2	4	PASS
3.3.4.2	D_1	PASS
3.3.4.2	5	PASS, NO_OPTION
3.3.4.4	6	PASS, NO_OPTION

Assertions for sigprocmask

Clause	Assertion ID	Conforming Results
3.3.5.2	D_1	PASS, NO_OPTION

Assertions for sigpending

Clause	Assertion ID	Conforming Results
No assertions for sigpending		

Assertions for sigsuspend

Clause	Assertion ID	Conforming Results
No assertions for sigsuspend		

Assertions for sigwaitinfo

Clause	Assertion ID	Conforming Results
3.3.8.1	1	PASS[1,2], NO_OPTION
3.3.8.1	2	PASS[1,2], NO_OPTION
3.3.8.1	3	PASS, NO_OPTION
3.3.8.1	4	PASS, NO_OPTION
3.3.8.2	9	PASS, NO_OPTION
3.3.8.2	10	PASS, NO_OPTION
3.3.8.2	D_1	PASS, NO_OPTION
3.3.8.2	11	PASS, NO_OPTION
3.3.8.2	12	PASS, NO_OPTION
3.3.8.2	13	PASS, NO_OPTION
3.3.8.2	14	PASS, NO_OPTION, NO_TEST
3.3.8.2	D_2	PASS, NO_OPTION
3.3.8.2	15	PASS, NO_OPTION
3.3.8.2	18	PASS, NO_OPTION
3.3.8.2	D_4	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.3.8.3	R_1	Reference Assertion, no test results
3.3.8.3	R_2	Reference Assertion, no test results
3.3.8.4	19	PASS, NO_OPTION
3.3.8.4	20	PASS, NO_OPTION
3.3.8.4	22	PASS, NO_OPTION
3.3.8.4	D_5	PASS, NO_OPTION

Assertions for sigtimedwait

Clause	Assertion ID	Conforming Results
3.3.8.1	5	PASS[5, 6], NO_OPTION
3.3.8.1	6	PASS[5, 6], NO_OPTION
3.3.8.1	7	PASS, NO_OPTION
3.3.8.1	8	PASS, NO_OPTION
3.3.8.1	8	PASS, NO_OPTION
3.3.8.2	10	PASS, NO_OPTION
3.3.8.2	D_1	PASS, NO_OPTION
3.3.8.2	11	PASS, NO_OPTION
3.3.8.2	12	PASS, NO_OPTION
3.3.8.2	13	PASS, NO_OPTION
3.3.8.2	14	PASS, NO_OPTION, NO_TEST
3.3.8.2	D_2	PASS, NO_OPTION
3.3.8.2	15	PASS, NO_OPTION
3.3.8.2	16	PASS, NO_OPTION
3.3.8.2	17	PASS, NO_OPTION
3.3.8.2	D_3	PASS, NO_OPTION
3.3.8.2	18	PASS, NO_OPTION
3.3.8.2	D_4	PASS, NO_OPTION
3.3.8.4	19	PASS, NO_OPTION
3.3.8.4	20	PASS, NO_OPTION
3.3.8.4	21	PASS, NO_OPTION

Assertions for sigqueue

Clause	Assertion ID	Conforming Results
3.3.9.1	1	PASS[1, 2], NO_OPTION
3.3.9.1	2	PASS[1, 2], NO_OPTION
3.3.9.1	3	PASS, NO_OPTION
3.3.9.1	4	PASS, NO_OPTION
3.3.9.2	5	PASS, NO_OPTION
3.3.9.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.9.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.9.2	8	PASS, NO_OPTION
3.3.9.2	9	PASS, NO_OPTION
3.3.9.2	10	PASS, NO_OPTION
3.3.9.2	11	PASS, NO_OPTION
3.3.9.2	D_1	PASS, NO_OPTION
3.3.9.2	12	PASS, NO_OPTION
3.3.9.2	D_2	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.3.9.3	R_1	Reference Assertion, no test results
3.3.9.3	R_2	Reference Assertion, no test results
3.3.9.4	13	PASS, NO_OPTION, NO_TEST
3.3.9.4	14	PASS, NO_OPTION
3.3.9.4	15	PASS, NO_OPTION, NO_TEST
3.3.9.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.9.4	17	PASS, NO_OPTION
3.3.9.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.9.4	19	PASS, NO_OPTION

Assertions for intro_timer_ops

Clause	Assertion ID	Conforming Results
No assertions for intro_timer_ops		

Assertions for alarm

Clause	Assertion ID	Conforming Results
No assertions for alarm		

Assertions for pause

Clause	Assertion ID	Conforming Results
No assertions for pause		

Assertions for sleep

Clause	Assertion ID	Conforming Results
No assertions for sleep		

A.4 Process Environment

Assertions for sysconf

Clause	Assertion ID	Conforming Results
4.8.1.2	1	PASS
4.8.1.2	2	PASS, NO_TEST_SUPPORT
4.8.1.2	3	PASS, NO_TEST_SUPPORT
4.8.1.2	4	PASS, NO_TEST_SUPPORT
4.8.1.2	5	PASS, NO_TEST_SUPPORT

A.5 Files and Directories

Assertions for intro5

Clause	Assertion ID	Conforming Results
No assertions for sleep		

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for dirent.h

Clause	Assertion ID	Conforming Results
No assertions for dirent.h		

Assertions for directory operations

Clause	Assertion ID	Conforming Results
No assertions for directory operations		

Assertions for chdir

Clause	Assertion ID	Conforming Results
No assertions for chdir		

Assertions for getcwd

Clause	Assertion ID	Conforming Results
No assertions for getcwd		

Assertions for open

Clause	Assertion ID	Conforming Results
5.3.1.2	GAsyncOpenWrite	General Assertion, no test results
5.3.1.4	1	PASS, NO_OPTION, NO_TEST_SUPPORT
5.3.1.5	2	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for creat

Clause	Assertion ID	Conforming Results
No assertions for creat		

Assertions for umask

Clause	Assertion ID	Conforming Results
No assertions for umask		

Assertions for link

Clause	Assertion ID	Conforming Results
No assertions for link		

Assertions for mkdir

Clause	Assertion ID	Conforming Results
No assertions for mkdir		

Assertions for mkfifo

Clause	Assertion ID	Conforming Results
No assertions for mkfifo		

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for unlink

Clause	Assertion ID	Conforming Results
No assertions for unlink		

Assertions for rmdir

Clause	Assertion ID	Conforming Results
No assertions for rmdir		

Assertions for rename

Clause	Assertion ID	Conforming Results
No assertions for rename		

Assertions for stat.h

Clause	Assertion ID	Conforming Results
5.6.1.1	D_1	PASS, NO_OPTION
5.6.1.1	1	PASS
5.6.1.1	2	PASS, NO_TEST_SUPPORT
5.6.1.1	3	PASS, NO_TEST_SUPPORT
5.6.1.1	4	PASS, NO_TEST_SUPPORT
5.6.1.1	5	PASS, NO_TEST_SUPPORT
5.6.1.1	6	PASS, NO_TEST_SUPPORT
5.6.1.1	7	PASS, NO_TEST_SUPPORT
5.6.1.1	8	PASS, NO_TEST_SUPPORT
5.6.1.1	9	PASS, NO_TEST_SUPPORT
5.6.1.1	10	PASS, NO_TEST_SUPPORT

Assertions for fstat

Clause	Assertion ID	Conforming Results
5.6.2.2	1	PASS

Assertions for stat

Clause	Assertion ID	Conforming Results
No assertions for stat		

Assertions for access

Clause	Assertion ID	Conforming Results
No assertions for access		

Assertions for fchmod

Clause	Assertion ID	Conforming Results
5.6.4.1	1	PASS[1, 2], NO_OPTION
5.6.4.1	2	PASS[1, 2], NO_OPTION
5.6.4.1	3	PASS, NO_OPTION
5.6.4.1	4	PASS, NO_OPTION
5.6.4.2	5	PASS, NO_OPTION
5.6.4.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
5.6.4.2	7	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.6.4.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
5.6.4.2	9	PASS, NO_OPTION
5.6.4.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
5.6.4.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
5.6.4.3	R_1	Reference Assertion, no test results
5.6.4.3	R_2	Reference Assertion, no test results
5.6.4.4	14	PASS, NO_OPTION
5.6.4.4	15	PASS, NO_OPTION
5.6.4.4	16	PASS, NO_OPTION
5.6.4.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT
5.6.4.4	18	PASS, NO_OPTION

Assertions for chmod

Clause	Assertion ID	Conforming Results
5.6.4.2	12	PASS, NO_TEST_SUPPORT

Assertions for utime

Clause	Assertion ID	Conforming Results
No assertions for utime		

Assertions for ftruncate

Clause	Assertion ID	Conforming Results
5.6.7.1	1	PASS[1, 2], NO_OPTION
5.6.7.1	2	PASS[1, 2], NO_OPTION
5.6.7.1	3	PASS, NO_OPTION
5.6.7.1	4	PASS, NO_OPTION
5.6.7.2	5	PASS, NO_OPTION
5.6.7.2	D_1	PASS, NO_OPTION
5.6.7.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
5.6.7.2	7	PASS, NO_OPTION
5.6.7.2	D_2	PASS, NO_OPTION
5.6.7.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
5.6.7.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT1
5.6.7.2	10	PASS, NO_OPTION
5.6.7.2	11	PASS, NO_OPTION
5.6.7.2	12	PASS, NO_OPTION
5.6.7.3	R_1	Reference Assertion, no test results
5.6.7.3	R_2	Reference Assertion, no test results
5.6.7.4	13	PASS, NO_OPTION
5.6.7.4	14	PASS, NO_OPTION
5.6.7.4	15	PASS, NO_OPTION

Assertions for pathconf

Clause	Assertion ID	Conforming Results
5.7.1.2	1	PASS

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.7.1.2	2	PASS, NO_OPTION
5.7.1.2	3	PASS
5.7.1.2	4	PASS, NO_OPTION
5.7.1.2	5	PASS
5.7.1.2	6	PASS, NO_OPTION
5.7.1.2	7	PASS
5.7.1.2	D_1	PASS, NO_OPTION

Assertions for fpathconf

Clause	Assertion ID	Conforming Results
5.7.1.2	2	PASS, NO_OPTION
5.7.1.2	3	PASS
5.7.1.2	4	PASS, NO_OPTION
5.7.1.2	5	PASS
5.7.1.2	6	PASS, NO_OPTION
5.7.1.2	7	PASS
5.7.1.2	D_1	PASS, NO_OPTION

A.6 Input and Output Primitives

Assertions for intro6

Clause	Assertion ID	Conforming Results
No assertions for intro6		

Assertions for pipe

Clause	Assertion ID	Conforming Results
No assertions for pipe		

Assertions for dup

Clause	Assertion ID	Conforming Results
No assertions for dup		

Assertions for dup2

Clause	Assertion ID	Conforming Results
No assertions for dup2		

Assertions for close

Clause	Assertion ID	Conforming Results
6.3.1.2	1	PASS, NO_OPTION, NO_TEST
6.3.1.2	2	PASS, NO_OPTION, NO_TEST
6.3.1.2	D_1	PASS
6.3.1.2	3	PASS, NO_OPTION
6.3.1.2	4	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Assertions for read

Clause	Assertion ID	Conforming Results
6.4.1.2	1	PASS, NO_OPTION, NO_TEST
6.4.1.2	2	PASS, NO_OPTION, NO_TEST
6.4.1.2	3	PASS, NO_OPTION, NO_TEST
6.4.1.2	D_1	PASS, NO_OPTIONS

Assertions for write

Clause	Assertion ID	Conforming Results
6.4.2.2	1	PASS, NO_OPTION, NO_TEST
6.4.2.2	2	PASS, NO_OPTION, NO_TEST
6.4.2.2	D_1	PASS, NO_OPTION

Assertions for fcntl.h

Clause	Assertion ID	Conforming Results
6.5.1	1	PASS

Assertions for fcntl

Clause	Assertion ID	Conforming Results
6.5.2.2	4	PASS
6.5.2.2	5	PASS
6.5.2.2	6	PASS
6.5.2.2	7	PASS
6.5.2.2	8	PASS
6.5.2.2	9	PASS
6.5.2.2	10	PASS
6.5.2.2	D_1	PASS, NO_OPTION
6.5.2.4	41	PASS
6.5.2.4	42	PASS, NO_TEST_SUPPORT
6.5.2.4	42.1	PASS, NO_TEST_SUPPORT
6.5.2.4	42.2	PASS, NO_OPTION
6.5.2.4	48	PASS, NO_TEST_SUPPORT
6.5.2.4	49	PASS, NO_TEST_SUPPORT
6.5.2.4	49.1	PASS, NO_TEST_SUPPORT

Assertions for lseek

Clause	Assertion ID	Conforming Results
6.5.3.2	D_1	PASS, NO_OPTION

Assertions for syncio

Clause	Assertion ID	Conforming Results
6.6	D_1	PASS

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for fsync

Clause	Assertion ID	Conforming Results
6.6.1.1	1	PASS[1, 2], NO_OPTION
6.6.1.1	2	PASS[1, 2], NO_OPTION
6.6.1.1	3	PASS, NO_OPTION
6.6.1.1	4	PASS, NO_OPTION
6.6.1.2	D_1	PASS, NO_OPTION
6.6.1.2	5	PASS, NO_OPTION, NO_TEST
6.6.1.2	R_1	Reference Assertion, no test results
6.6.1.2	D_2	PASS, NO_OPTION
6.6.1.2	6	PASS, NO_OPTION, NO_TEST
6.6.1.2	7	PASS, NO_OPTION, NO_TEST
6.6.1.3	R_2	Reference Assertion, no test results
6.6.1.3	8	PASS
6.6.1.4	9	PASS, NO_OPTION
6.6.1.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
6.6.1.4	11	PASS, NO_OPTION
6.6.1.4	12	PASS, NO_OPTION, NO_TEST

Assertions for fdatasync

Clause	Assertion ID	Conforming Results
6.6.2.1	1	PASS[1,2], NO_OPTION
6.6.2.1	2	PASS[1,2], NO_OPTION
6.6.2.1	3	PASS, NO_OPTION
6.6.2.1	4	PASS, NO_OPTION
6.6.2.2	5	PASS, NO_OPTION, NO_TEST
6.6.2.2	R_1	Reference Assertion, no test results
6.6.2.2	6	PASS, NO_OPTION, NO_TEST
6.6.2.2	7	PASS, NO_OPTION, NO_TEST
6.6.2.3	8	PASS
6.6.2.4	9	PASS, NO_OPTION
6.6.2.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
6.6.2.4	11	PASS, NO_OPTION
6.6.2.4	12	PASS, NO_OPTION, NO_TEST

Assertions for aio.h

Clause	Assertion ID	Conforming Results
6.7.1.1	1	PASS
6.7.1.1	D_1	PASS
6.7.1.1	2	PASS, NO_TEST
6.7.1.1	3	PASS, NO_OPTION
6.7.1.1	4	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.1.1	D_2	PASS
6.7.1.1	5	PASS, NO_OPTION
6.7.1.1	6	PASS, NO_OPTION
6.7.1.1	7	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.1.1	8	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.1.1	D_3	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.7.1.1	9	PASS, NO_OPTION
6.7.1.1	10	PASS, NO_OPTION
6.7.1.1	D_4	PASS
6.7.1.1	11	PASS, NO_OPTION
6.7.1.1	12	PASS, NO_OPTION
6.7.1.1	13	PASS, NO_OPTION
6.7.1.1	14	PASS, NO_OPTION
6.7.1.2	15	PASS
6.7.1.2	16	PASS
6.7.1.2	17	PASS

Assertions for aio_read

Clause	Assertion ID	Conforming Results
6.7.2.1	1	PASS[1, 2], NO_OPTION
6.7.2.1	2	PASS[1, 2], NO_OPTION
6.7.2.1	3	PASS, NO_OPTION
6.7.2.1	4	PASS, NO_OPTION
6.7.2.2	5	PASS, NO_OPTION
6.7.2.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.2.2	7	PASS, NO_OPTION, NO_TEST
6.7.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.2.2	9	PASS, NO_OPTION, NO_TEST
6.7.2.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.2.2	11	PASS, NO_OPTION, NO_TEST
6.7.2.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.2.2	R_1	Reference Assertion, no test results
6.7.2.2	13	PASS, NO_OPTION
6.7.2.2	D_1	PASS, NO_OPTION
6.7.2.2	14	PASS, NO_OPTION
6.7.2.2	D_2	PASS, NO_OPTION
6.7.2.2	D_3	PASS, NO_OPTION
6.7.2.2	15	PASS, NO_OPTION, NO_TEST
6.7.2.2	16	PASS, NO_OPTION, NO_TEST
6.7.2.2	17	PASS, NO_OPTION, NO_TEST
6.7.2.2	D_4	PASS, NO_OPTION
6.7.2.3	R_2	Reference Assertion, no test results
6.7.2.3	R_3	Reference Assertion, no test results
6.7.2.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.2.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.2.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.2.4	21	PASS, NO_OPTION
6.7.2.4	ebadf1	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.2.4	ebadf2	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.2.4	einval1	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.2.4	22	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.2.4	R_4	Reference Assertion, no test results
6.7.2.4	R_5	Reference Assertion, no test results

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.7.2.4	23	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.2.4	R_6	Reference Assertion, no test results

Assertions for aio_write

Clause	Assertion ID	Conforming Results
6.7.3.1	1	PASS[1, 2], NO_OPTION
6.7.3.1	2	PASS[1, 2], NO_OPTION
6.7.3.1	3	PASS, NO_OPTION
6.7.3.1	4	PASS, NO_OPTION
6.7.3.2	5	PASS, NO_OPTION
6.7.3.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.2	7	PASS, NO_OPTION, NO_TEST
6.7.3.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.2	9	PASS, NO_OPTION, NO_TEST
6.7.3.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.2	11	PASS, NO_OPTION, NO_TEST
6.7.3.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.2	D_1	PASS, NO_OPTION
6.7.3.2	13	PASS, NO_OPTION
6.7.3.2	14	PASS, NO_OPTION
6.7.3.2	D_2	PASS, NO_OPTION
6.7.3.2	15	PASS, NO_OPTION
6.7.3.2	D_3	PASS, NO_OPTION
6.7.3.2	16	PASS, NO_OPTION, NO_TEST
6.7.3.2	17	PASS, NO_OPTION, NO_TEST
6.7.3.2	D_4	PASS, NO_OPTION
6.7.3.3	R_1	Reference Assertion, no test results
6.7.3.3	R_2	Reference Assertion, no test results
6.7.3.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.3.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.3.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.4	21	PASS, NO_OPTION
6.7.3.4	ebadf1	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.4	ebadf2	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.4	einval1	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.4	22	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.4	23	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.4	24	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.4	25	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.4	26	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.4	27	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.4	R_3	Reference Assertion, no test results
6.7.3.4	R_4	Reference Assertion, no test results
6.7.3.4	28	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.3.4	R_5	Reference Assertion, no test results

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for lio_listio

Clause	Assertion ID	Conforming Results
6.7.4.1	1	PASS[1, 2], NO_OPTION
6.7.4.1	2	PASS[1, 2], NO_OPTION
6.7.4.1	3	PASS, NO_OPTION
6.7.4.1	4	PASS, NO_OPTION
6.7.4.2	5	PASS, NO_OPTION
6.7.4.2	6	PASS, NO_OPTION
6.7.4.2	7	PASS, NO_OPTION
6.7.4.2	8	PASS, NO_OPTION
6.7.4.2	D_1	PASS, NO_OPTION
6.7.4.2	9	PASS, NO_OPTION
6.7.4.2	10	PASS, NO_OPTION
6.7.4.2	lio_read_op	PASS, NO_OPTION
6.7.4.2	lio_write_op	PASS, NO_OPTION
6.7.4.2	R_1	Reference Assertion, no test results
6.7.4.2	11	PASS, NO_OPTION, NO_TEST
6.7.4.2	12	PASS, NO_OPTION, NO_TEST
6.7.4.2	13	PASS, NO_OPTION, NO_TEST
6.7.4.2	14	PASS, NO_OPTION, NO_TEST
6.7.4.3	R_2	Reference Assertion, no test results
6.7.4.3	R_3	Reference Assertion, no test results
6.7.4.3	R_4	Reference Assertion, no test results
6.7.4.3	R_5	Reference Assertion, no test results
6.7.4.3	15	PASS, NO_OPTION
6.7.4.3	R_6	Reference Assertion, no test results
6.7.4.4	16	PASS, NO_OPTION, NO_TEST
6.7.4.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.4.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	lio_read_ebadf1	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	lio_read_ebadf2	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	lio_read_einval1	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	21	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	R_7	Reference Assertion, no test results
6.7.4.4	R_8	Reference Assertion, no test results
6.7.4.4	22	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	R_9	Reference Assertion, no test results
6.7.4.4	lio_write_ebadf1	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	lio_write_ebadf2	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	lio_write_einval1	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	23	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	24	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	25	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	26	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	27	PASS, NO_OPTION, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.7.4.4	R_10	Reference Assertion, no test results
6.7.4.4	R_11	Reference Assertion, no test results
6.7.4.4	28	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	29	PASS, NO_OPTION
6.7.4.4	30	PASS, NO_OPTION
6.7.4.4	31	PASS, NO_OPTION
6.7.4.4	32	PASS, NO_OPTION

Assertions for aio_error

Clause	Assertion ID	Conforming Results
6.7.5.1	1	PASS[1, 2], NO_OPTION
6.7.5.1	2	PASS[1, 2], NO_OPTION
6.7.5.1	3	PASS, NO_OPTION
6.7.5.1	4	PASS, NO_OPTION
6.7.5.2	5	PASS, NO_OPTION
6.7.5.2	R_1	Reference Assertion, no test results
6.7.5.2	6	PASS, NO_OPTION
6.7.5.3	7	PASS, NO_OPTION
6.7.5.3	R_2	Reference Assertion, no test results
6.7.5.3	8	PASS, NO_OPTION
6.7.5.4	9	PASS, NO_OPTION
6.7.5.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.5.4	11	PASS, NO_OPTION

Assertions for aio_return

Clause	Assertion ID	Conforming Results
6.7.6.1	1	PASS[1, 2], NO_OPTION
6.7.6.1	2	PASS[1, 2], NO_OPTION
6.7.6.1	3	PASS, NO_OPTION
6.7.6.1	4	PASS, NO_OPTION
6.7.6.2	return_status	PASS, NO_OPTION
6.7.6.2	R_1	Reference Assertion, no test results
6.7.6.2	D_1	PASS, NO_OPTION
6.7.6.2	R_2	Refernce Assertion, no test results
6.7.6.2	5	PASS, NO_OPTION
6.7.6.3	R_3	Reference Assertion, no test results
6.7.6.3	D_2	PASS, NO_OPTION
6.7.6.4	einval1	PASS, NO_OPTION
6.7.6.4	6	PASS, NO_OPTION
6.7.6.4	7	PASS, NO_OPTION

Assertions for aio_cancel

Clause	Assertion ID	Conforming Results
6.7.7.1	1	PASS[1, 2], NO_OPTION
6.7.7.1	2	PASS[1, 2], NO_OPTION
6.7.7.1	3	PASS, NO_OPTION
6.7.7.1	4	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.7.7.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.7.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.2	D_1	PASS, NO_OPTION
6.7.7.2	D_2	PASS, NO_OPTION
6.7.7.3	R_1	Reference Assertion, no test results
6.7.7.3	10	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.7.3	11	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.3	R_3	Reference Assertion, no test results
6.7.7.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.4	13	PASS, NO_OPTION

Assertions for aio_suspend

Clause	Assertion ID	Conforming Results
6.7.8.1	1	PASS[1, 2], NO_OPTION
6.7.8.1	2	PASS[1, 2], NO_OPTION
6.7.8.1	3	PASS, NO_OPTION
6.7.8.1	4	PASS, NO_OPTION
6.7.8.2	completion	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.8.2	interrupt	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.8.2	timeout	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.8.2	already_completed	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.8.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.8.2	D_1	PASS, NO_OPTION
6.7.8.2	R_1	Reference Assertion, no test results
6.7.8.3	R_2	Reference Assertion, no test results
6.7.8.3	R_3	Reference Assertion, no test results
6.7.8.4	R_4	Reference Assertion, no test results
6.7.8.4	6	PASS
6.7.8.4	no_support	PASS, NO_OPTION

Assertions of aio_fsync

Clause	Assertion ID	Conforming Results
6.7.9.1	1	PASS[1, 2], NO_OPTION
6.7.9.1	2	PASS[1, 2], NO_OPTION
6.7.9.1	3	PASS, NO_OPTION
6.7.9.1	4	PASS, NO_OPTION
6.7.9.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.9.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.9.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.9.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.9.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.9.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.7.9.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.2	D_1	PASS, NO_OPTION
6.7.9.3	R_1	Reference Assertion, no test results
6.7.9.3	R_2	Reference Assertion, no test results
6.7.9.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.4	15	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.4	18	PASS, NO_OPTION
6.7.9.4	19	PASS, NO_OPTION

A.7 Device- and Class-Specific Functions

Assertions for Section_7

Clause	Assertion ID	Conforming Results
No assertions for Section_7		

A.8 Language-Specific Services for the C Programming Language

Clause	Assertion ID	Conforming Results
8.2.2.2	D_1	PASS, NO_OPTION

A.9 System Databases

Assertions for Section_9

Clause	Assertion ID	Conforming Results
No assertions for Section_9		

A.10 Data Interchange Format

Assertions for Section_10

Clause	Assertion ID	Conforming Results
No assertions for Section_10		

A.11 Synchronization

Assertions for sem_hdr

Clause	Assertion ID	Conforming Results
11.1	1	PASS
11.1	GA_semOpenMaxFD	General Assertion, no test results
11.1	GA_semPCTSOpenMaxFD	General Assertion, no test results

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Assertions for sem_init

Clause	Assertion ID	Conforming Results
11.2.1.1	1	PASS[1, 2], NO_OPTION
11.2.1.1	2	PASS[1, 2], NO_OPTION
11.2.1.1	3	PASS, NO_OPTION
11.2.1.1	4	PASS, NO_OPTION
11.2.1.2	OpenMaxSems	PASS[OpenMaxSems, PCTSOpenMaxSems]
11.2.1.2	PCTS OpenMaxSems	PASS[OpenMaxSems, PCTSOpenMaxSems]
11.2.1.2	sem_init	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.2	D_1	PASS, NO_OPTION
11.2.1.2	D_2	PASS, NO_OPTION
11.2.1.2	D_3	PASS, NO_OPTION
11.2.1.3	R_1	Reference Assertion, no test results
11.2.1.3	R_2	Reference Assertion, no test results
11.2.1.4	8	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	9	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
11.2.1.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	11	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	15	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for sem-destroy

Clause	Assertion ID	Conforming Results
11.2.2.1	1	PASS[1, 2], NO_OPTION
11.2.1.1	2	PASS[1, 2], NO_OPTION
11.2.2.1	3	PASS, NO_OPTION
11.2.2.1	4	PASS, NO_OPTION
11.2.2.2	sem_destroy	PASS, NO_OPTION
11.2.2.2	D_1	PASS, NO_OPTION
11.2.2.2	D_2	PASS, NO_OPTION
11.2.2.2	D_3	PASS, NO_OPTION
11.2.2.2	D_4	PASS, NO_OPTION
11.2.2.3	R_1	Reference Assertion, no test results
11.2.2.3	R_2	Reference Assertion, no test results
11.2.2.4	5	PASS, NO_OPTION
11.2.2.4	6	PASS, NO_OPTION
11.2.2.4	7	PASS, NO_OPTION

Assertions for sem_open

Clause	Assertion ID	Conforming Results
11.2.3.1	1	PASS[1, 2], NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.2.3.1	2	PASS[1, 2], NO_OPTION
11.2.3.1	3	PASS, NO_OPTION
11.2.3.1	4	PASS, NO_OPTION
11.2.3.2	OpenMaxSems	PASS[OpenMaxSems, PCTSOpenMaxSems]
11.2.3.2	PCTSOpenMaxSems	PASS[OpenMaxSems, PCTSOpenMaxSems]
11.2.3.2	sem_open	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.3.2	5	PASS, NO_OPTION
11.2.3.2	6	PASS, NO_OPTION
11.2.3.2	7	PASS, NO_OPTION
11.2.3.2	8	PASS, NO_OPTION
11.2.3.2	9	PASS, NO_OPTION
11.2.3.2	10	PASS, NO_OPTION
11.2.3.2	11	PASS, NO_OPTION
11.2.3.2	12	PASS, NO_OPTION
11.2.3.2	D_1	PASS, NO_OPTION
11.2.3.2	13	PASS, NO_OPTION
11.2.3.2	14	PASS, NO_OPTION
11.2.3.2	15	PASS, NO_OPTION, NO_TEST
11.2.3.2	D_2	PASS, NO_OPTION
11.2.3.2	D_3	PASS, NO_OPTION
11.2.3.2	D_4	PASS, NO_OPTION
11.2.3.2	D_5	PASS, NO_OPTION
11.2.3.2	16	PASS, NO_OPTION
11.2.3.2	17	PASS, NO_OPTION
11.2.3.2	18	PASS, NO_OPTION
11.2.3.2	19	PASS, NO_OPTION
11.2.3.2	20	PASS, NO_OPTION
11.2.3.2	D_6	PASS, NO_OPTION
11.2.3.2	D_7	PASS, NO_OPTION
11.2.3.2	21	PASS, NO_OPTION
11.2.3.2	D_8	PASS, NO_OPTION
11.2.3.2	D_9	PASS, NO_OPTION
11.2.3.3	R_1	Reference Assertion, no test results
11.2.3.4	22	PASS, NO_OPTION
11.2.3.4	23	PASS, NO_OPTION
11.2.3.4	24	PASS, NO_OPTION
11.2.3.4	25	PASS, NO_OPTION
11.2.3.4	D_10	PASS, NO_OPTION
11.2.3.4	26	PASS, NO_OPTION
11.2.3.4	27	PASS, NO_OPTION
11.2.3.4	28	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.3.4	29	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.3.4	30	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.3.4	31	PASS, NO_OPTION
11.2.3.4	32	PASS, NO_OPTION
11.2.3.4	33	PASS, NO_OPTION, NO_TEST
11.2.3.4	34	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for sem_close

Clause	Assertion ID	Conforming Results
11.2.4.1	1	PASS[1, 2], NO_OPTION
11.2.4.1	2	PASS[1, 2], NO_OPTION
11.2.4.1	3	PASS, NO_OPTION
11.2.4.1	4	PASS, NO_OPTION
11.2.4.2	sem_close	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.4.2	D_1	PASS, NO_OPTION
11.2.4.2	D_2	PASS, NO_OPTION
11.2.4.2	5	PASS, NO_OPTION
11.2.4.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.4.2	D_3	PASS, NO_OPTION
11.2.4.3	R_1	Reference Assertion, no test results
11.2.4.3	R_2	Reference Assertion, no test results
11.2.4.4	7	PASS, NO_OPTION
11.2.4.4	8	PASS, NO_OPTION

Assertions for sem_unlink

Clause	Assertion ID	Conforming Results
11.2.5.1	1	PASS[1, 2], NO_OPTION
11.2.5.1	2	PASS[1, 2], NO_OPTION
11.2.5.1	3	PASS, NO_OPTION
11.2.5.1	4	PASS, NO_OPTION
11.2.5.2	sem_unlink	PASS, NO_OPTION
11.2.5.2	5	PASS, NO_OPTION
11.2.5.2	6	PASS, NO_OPTION
11.2.5.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.5.2	8	PASS, NO_OPTION
11.2.5.2	D_1	PASS, NO_OPTION
11.2.5.3	R_1	Reference Assertion, no test results
11.2.5.3	R_2	Reference Assertion, no test results
11.2.5.4	9	PASS, NO_OPTION
11.2.5.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.5.4	11	PASS, NO_OPTION
11.2.5.4	12	PASS, NO_OPTION

Assertions for sem_wait

Clause	Assertion ID	Conforming Results
11.2.6.1	1	PASS[1, 2], NO_OPTION
11.2.6.1	2	PASS[1, 2], NO_OPTION
11.2.6.1	3	PASS, NO_OPTION
11.2.6.1	4	PASS, NO_OPTION
11.2.6.2	sem_wait	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	14	PASS, NO_OPTION, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.2.6.2	D_1	PASS, NO_OPTION
11.2.6.3	15	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.3	R_1	Reference Assertion, no test results
11.2.6.3	R_3	Reference Assertion, no test results
11.2.6.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	21	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	23	PASS, NO_OPTION
11.2.6.4	25	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for sem_trywait

Clause	Assertion ID	Conforming Results
11.2.6.1	5	PASS[5, 6], NO_OPTION
11.2.6.1	6	PASS[5, 6], NO_OPTION
11.2.6.1	7	PASS, NO_OPTION
11.2.6.1	8	PASS, NO_OPTION
11.2.6.2	sem_trywait	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	D_2	PASS, NO_OPTION
11.2.6.3	16	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.3	R_2	Reference Assertion, no test results
11.2.6.3	R_4	Reference Assertion, no test results
11.2.6.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	22	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	24	PASS, NO_OPTION
11.2.6.4	26	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for sem_post

Clause	Assertion ID	Conforming Results
11.2.7.1	1	PASS[1, 2], NO_OPTION
11.2.7.1	2	PASS[1, 2], NO_OPTION
11.2.7.1	3	PASS, NO_OPTION
11.2.7.1	4	PASS, NO_OPTION
11.2.7.2	sem_post	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	R_1	Reference Assertion, no test results
11.2.7.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	D_1	PASS, NO_OPTION
11.2.7.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	D_2	PASS, NO_OPTION
11.2.7.3	R_2	Reference Assertion, no test results

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.2.7.3	R_3	Reference Assertion, no test results
11.2.7.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.4	11	PASS, NO_OPTION

Assertions for sem_getvalue

Clause	Assertion ID	Conforming Results
11.2.8.1	1	PASS[1, 2], NO_OPTION
11.2.8.1	2	PASS[1, 2], NO_OPTION
11.2.8.1	3	PASS, NO_OPTION
11.2.8.1	4	PASS, NO_OPTION
11.2.8.2	sem_getvalue	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.8.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.8.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.8.2	D_1	PASS, NO_OPTION
11.2.8.3	R_1	Reference Assertion, no test results
11.2.8.3	R_2	Reference Assertion, no test results
11.2.8.4	7	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.8.4	8	PASS, NO_OPTION

A.12 Memory Management

Assertions for mem-intro

Clause	Assertion ID	Conforming Results
12	1	PASS
12	D_1	PASS, NO_OPTION
12	D_2	PASS, NO_OPTION
12	D_3	PASS, NO_OPTION
12	R_1	Reference Assertion, no test results
12	R_2	Reference Assertion, no test results
12	R_3	Reference Assertion, no test results
12	2	PASS, NO_TEST_SUPPORT
12	3	PASS, NO_TEST_SUPPORT
12	4	PASS, NO_TEST_SUPPORT
12	R_4	Reference Assertion, no test results
12	R_5	Reference Assertion, no test results
12	D_4	PASS, NO_OPTION
12	5	PASS, NO_OPTION, NO_TEST_SUPPORT
12	6	PASS, NO_OPTION, NO_TEST_SUPPORT
12	7	PASS, NO_OPTION, NO_TEST_SUPPORT
12	R_6	Reference Assertion, no test results
12	D_5	PASS, NO_OPTION

Assertions for mlockall

Clause	Assertion ID	Conforming Results
12.1.1.1	1	PASS[1, 2], NO_OPTION
12.1.1.1	2	PASS[1, 2], NO_OPTION
12.1.1.1	3	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12.1.1.1	4	PASS, NO_OPTION
12.1.1.2	mlockall	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.1.2	9	PASS, NO_OPTION
12.1.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.1.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.1.2	D_1	PASS, NO_OPTION
12.1.1.2	14	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.1.2	D_2	PASS, NO_OPTION
12.1.1.2	15	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.1.2	D_3	PASS, NO_OPTION
12.1.1.2	D_4	PASS, NO_OPTION
12.1.1.3	R_1	Reference Assertion, no test results
12.1.1.3	R_2	Reference Assertion, no test results
12.1.1.3	D_5	PASS, NO_OPTION
12.1.1.4	16	PASS, NO_OPTION
12.1.1.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.1.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.1.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.1.4	21	PASS, NO_OPTION
12.1.1.4	D_6	PASS, NO_OPTION
12.1.1.4	22	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for munlockall

Clause	Assertion ID	Conforming Results
12.1.1.1	5	PASS[5, 6], NO_OPTION
12.1.1.1	6	PASS[5, 6], NO_OPTION
12.1.1.1	7	PASS, NO_OPTION
12.1.1.1	8	PASS, NO_OPTION
12.1.1.2	munlockall	PASS, NO_OPTION
12.1.1.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.1.2	13	PASS, NO_OPTION
12.1.1.3	R_3	Reference Assertion, no test results
12.1.1.4	17	PASS, NO_OPTION

Assertions for mlock

Clause	Assertion ID	Conforming Results
12.1.2.1	1	PASS[1, 2], NO_OPTION
12.1.2.1	2	PASS[1, 2], NO_OPTION
12.1.2.1	3	PASS, NO_OPTION
12.1.2.1	4	PASS, NO_OPTION
12.1.2.2	mlock	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.2.2	D_1	PASS, NO_OPTION
12.1.2.2	D_3	PASS, NO_OPTION
12.1.2.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.2.2	D_4	PASS, NO_OPTION
12.1.2.2	D_5	PASS, NO_OPTION
12.1.2.3	R_1	Reference Assertion, no test results

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12.1.2.3	R_2	Reference Assertion, no test results
12.1.2.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.2.4	14	PASS, NO_OPTION
12.1.2.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.2.4	D_6	PASS, NO_OPTION
12.1.2.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.2.4	D_7	PASS, NO_OPTION
12.1.2.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.2.4	D_8	PASS, NO_OPTION
12.1.2.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for munlock

Clause	Assertion ID	Conforming Results
12.1.2.1	5	PASS[5, 6], NO_OPTION
12.1.2.1	6	PASS[5, 6], NO_OPTION
12.1.2.1	7	PASS, NO_OPTION
12.1.2.1	8	PASS, NO_OPTION
12.1.2.2	munlock	PASS, NO_OPTION, NO_TEST_SUPPORT
12.1.2.2	D_2	PASS, NO_OPTION
12.1.2.2	9	PASS, NO_OPTION
12.1.2.2	10	PASS, NO_OPTION
12.1.2.3	R_3	Reference Assertion, no test results
12.1.2.4	13	PASS, NO_OPTION
12.1.2.4	15	PASS, NO_OPTION
12.1.2.4	18	PASS, NO_OPTION

Assertions for mmap

Clause	Assertion ID	Conforming Results
12.2.1.1	1	PASS[1, 2], NO_OPTION
12.2.1.1	2	PASS[1, 2], NO_OPTION
12.2.1.1	3	PASS, NO_OPTION
12.2.1.1	4	PASS, NO_OPTION
12.2.1.2	mmap	PASS, NO_OPTION
12.2.1.2	D_1	PASS, NO_OPTION
12.2.1.2	5	PASS, NO_OPTION
12.2.1.2	6	PASS, NO_OPTION
12.2.1.2	7	PASS, NO_OPTION
12.2.1.2	8	PASS, NO_OPTION
12.2.1.2	prot_values	PASS, NO_OPTION
12.2.1.2	9	PASS, NO_OPTION
12.2.1.2	R_1	Reference Assertion, no test results
12.2.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	mem_protect_flags	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	12	PASS, NO_OPTION
12.2.1.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	D_2	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12.2.1.2	14	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	15	PASS, NO_OPTION
12.2.1.2	16	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	D_3	PASS, NO_OPTION
12.2.1.2	D_4	PASS, NO_OPTION
12.2.1.2	17	PASS, NO_OPTION
12.2.1.2	18	PASS, NO_OPTION
12.2.1.2	19	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	20	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	D_5	PASS, NO_OPTION
12.2.1.2	21	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	D_6	PASS, NO_OPTION
12.2.1.2	22	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	23	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	24	PASS, NO_OPTION
12.2.1.2	25	PASS, NO_OPTION
12.2.1.2	mmap_SIGBUS	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.2	D_7	PASS, NO_OPTION
12.2.1.2	D_8	PASS, NO_OPTION
12.2.1.3	R_2	Reference Assertion, no test results
12.2.1.3	R_3	Reference Assertion, no test results
12.2.1.3	26	PASS, NO_OPTION
12.2.1.3	R_4	Reference Assertion, no test results
12.2.1.4	27	PASS, NO_OPTION
12.2.1.4	28	PASS, NO_OPTION
12.2.1.4	29	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	30	PASS, NO_OPTION
12.2.1.4	31	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	32	PASS, NO_OPTION
12.2.1.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
12.2.1.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	35	PASS, NO_OPTION
12.2.1.4	36	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	37	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	mmap_ENOTSUP	PASS, NO_OPTION
12.2.1.4	38	PASS, NO_OPTION
12.2.1.4	39	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	40	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	41	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for munmap

Clause	Assertion ID	Conforming Results
12.2.2.1	1	PASS[1, 2], NO_OPTION
12.2.2.1	2	PASS[1, 2], NO_OPTION
12.2.2.1	3	PASS, NO_OPTION
12.2.2.1	4	PASS, NO_OPTION
12.2.2.2	munmap	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12.2.2.2	munmap_SIGSEGV	PASS, NO_OPTION
12.2.2.2	5	PASS, NO_OPTION
12.2.2.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.2.2	D_1	PASS, NO_OPTION
12.2.2.2	munlock_remove_maps	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.2.2	D_2	PASS, NO_OPTION
12.2.2.2	D_3	PASS, NO_OPTION
12.2.2.3	R_1	Reference Assertion, no test results
12.2.2.3	R_2	Reference Assertion, no test results
12.2.2.4	8	PASS, NO_OPTION, NO_TEST
12.2.2.4	9	PASS, NO_OPTION
12.2.2.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for mprotect

Clause	Assertion ID	Conforming Results
12.2.3.1	1	PASS[1, 2], NO_OPTION
12.2.3.1	2	PASS[1, 2], NO_OPTION
12.2.3.1	3	PASS, NO_OPTION
12.2.3.1	4	PASS, NO_OPTION
12.2.3.2	mprotect	PASS, NO_OPTION
12.2.3.2	R_1	Reference Assertion, no test results
12.2.3.2	5	PASS, NO_OPTION
12.2.3.2	R_2	Reference Assertion, no test results
12.2.3.2	D_1	PASS, NO_OPTION
12.2.3.2	6	PASS, NO_OPTION
12.2.3.2	7	PASS, NO_OPTION
12.2.3.2	8	PASS, NO_OPTION
12.2.3.2	9	PASS, NO_OPTION
12.2.3.2	D_2	PASS, NO_OPTION
12.2.3.2	D_3	PASS, NO_OPTION
12.2.3.2	D_4	PASS, NO_OPTION
12.2.3.3	R_3	Reference Assertion, no test results
12.2.3.3	R_4	Reference Assertion, no test results
12.2.3.3	10	PASS, NO_OPTION
12.2.3.4	11	PASS, NO_OPTION
12.2.3.4	12	PASS, NO_OPTION
12.2.3.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TESTS
12.2.3.4	14	PASS, NO_OPTION
12.2.3.4	15	PASS, NO_OPTION
12.2.3.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.3.4	17	PASS, NO_OPTION
12.2.3.4	mprotect_ENOTSUP	PASS, NO_OPTION
12.2.3.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for msync

Clause	Assertion ID	Conforming Results
12.2.4.1	1	PASS[1, 2], NO_OPTION
12.2.4.1	2	PASS[1, 2], NO_OPTION
12.2.4.1	3	PASS, NO_OPTION
12.2.4.1	4	PASS, NO_OPTION
12.2.4.2	msync	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.4.2	5	PASS, NO_OPTION, NO_TEST
12.2.4.2	D_1	PASS, NO_OPTION
12.2.4.2	D_2	PASS, NO_OPTION
12.2.4.2	D_3	PASS, NO_OPTION
12.2.4.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.4.2	D_4	PASS, NO_OPTION
12.2.4.2	D_5	PASS, NO_OPTION
12.2.4.2	7	PASS, NO_OPTION
12.2.4.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
12.2.4.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
12.2.4.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
12.2.4.2	R_1	Reference Assertion, no test results
12.2.4.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
12.2.4.2	D_6	PASS, NO_OPTION
12.2.4.2	D_7	PASS, NO_OPTION
12.2.4.3	R_2	Reference Assertion, no test results
12.2.4.3	R_3	Reference Assertion, no test results
12.2.4.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.4.4	msync_einval	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.4.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.4.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.4.4	15	PASS, NO_OPTION
12.2.4.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for shm_open

Clause	Assertion ID	Conforming Results
12.3.1.1	1	PASS[1, 2], NO_OPTION
12.3.1.1	2	PASS[1, 2], NO_OPTION
12.3.1.1	3	PASS, NO_OPTION
12.3.1.1	4	PASS, NO_OPTION
12.3.1.2	shm_open	PASS, NO_OPTION
12.3.1.2	5	PASS, NO_OPTION
12.3.1.2	D_1	PASS, NO_OPTION
12.3.1.2	6	PASS, NO_OPTION
12.3.1.2	7	PASS, NO_OPTION
12.3.1.2	8	PASS, NO_OPTION
12.3.1.2	9	PASS, NO_OPTION
12.3.1.2	10	PASS, NO_OPTION
12.3.1.2	D_2	PASS, NO_OPTION
12.3.1.2	D_3	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12.3.1.2	11	PASS, NO_OPTION
12.3.1.2	12	PASS, NO_OPTION
12.3.1.2	D_4	PASS, NO_OPTION
12.3.1.2	13	PASS, NO_OPTION
12.3.1.2	14	PASS, NO_OPTION
12.3.1.2	15	PASS, NO_OPTION
12.3.1.2	16	PASS, NO_OPTION
12.3.1.2	17	PASS, NO_OPTION
12.3.1.2	18	PASS, NO_OPTION
12.3.1.2	19	PASS, NO_OPTION
12.3.1.2	20	PASS, NO_OPTION
12.3.1.2	21	PASS, NO_OPTION
12.3.1.2	22	PASS, NO_OPTION
12.3.1.2	23	PASS, NO_OPTION
12.3.1.2	24	PASS, NO_OPTION
12.3.1.2	D_5	PASS, NO_OPTION
12.3.1.2	25	PASS, NO_OPTION
12.3.1.2	26	PASS, NO_OPTION
12.3.1.2	R_1	Reference Assertion, no test results
12.3.1.2	27	PASS, NO_OPTION, NO_TEST
12.3.1.2	D_6	PASS, NO_OPTION
12.3.1.2	28	PASS, NO_OPTION
12.3.1.2	29	PASS, NO_OPTION
12.3.1.2	D_7	PASS, NO_OPTION
12.3.1.2	30	PASS, NO_OPTION
12.3.1.2	D_8	PASS, NO_OPTION
12.3.1.2	D_9	PASS, NO_OPTION
12.3.1.3	R_2	Reference Assertion, no test results
12.3.1.3	R_3	Reference Assertion, no test results
12.3.1.4	31	PASS, NO_OPTION
12.3.1.4	32	PASS, NO_OPTION
12.3.1.4	33	PASS, NO_OPTION
12.3.1.4	34	PASS, NO_OPTION
12.3.1.4	35	PASS, NO_OPTION
12.3.1.4	D_10	PASS, NO_OPTION
12.3.1.4	36	PASS, NO_OPTION
12.3.1.4	37	PASS, NO_OPTION, NO_TEST_SUPPORT
12.3.1.4	38	PASS, NO_OPTION, NO_TEST_SUPPORT
12.3.1.4	39	PASS, NO_OPTION
12.3.1.4	40	PASS, NO_OPTION
12.3.1.4	41	PASS, NO_OPTION, NO_TEST
12.3.1.4	42	PASS, NO_OPTION

Assertions for shm_unlink

Clause	Assertion ID	Conforming Results
12.3.2.1	1	PASS[1, 2], NO_OPTION
12.3.2.1	2	PASS[1, 2], NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12.3.2.1	3	PASS, NO_OPTION
12.3.2.1	4	PASS, NO_OPTION
12.3.2.2	shm_unlink	PASS, NO_OPTION
12.3.2.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
12.3.2.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
12.3.2.2	D_1	PASS, NO_OPTION
12.3.2.3	R_1	Reference Assertion, no test results
12.3.2.3	R_2	Reference Assertion, no test results
12.3.2.4	7	PASS, NO_OPTION
12.3.2.4	8	PASS, NO_OPTION, NO_TEST_RESULTS
12.3.2.4	9	PASS, NO_OPTION
12.3.2.4	10	PASS, NO_OPTION

A.13 Execution Scheduling

Assertions for sched_param

Clause	Assertion ID	Conforming Results
13.1	1	PASS
13.1	D_1	PASS, NO_OPTION
13.1	2	PASS, NO_TEST
13.1	3	PASS

Assertions for sched_policy

Clause	Assertion ID	Conforming Results
13.2	1	PASS
13.2	2	PASS, NO_TEST
13.2	D_1	PASS, NO_OPTION
13.2	3	PASS
13.2.1	sched_fifo1	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.1	sched_fifo2	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
13.2.1	sched_fifo3	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.1	sched_fifo4	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.1	sched_fifo5	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.1	sched_fifo6	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.1	sched_fifo7	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.1	sched_fifo8	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.1	R_1	Reference Assertion, no test results
13.2.1	4	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.1	5	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
13.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
13.2.3	D_2	PASS
13.2.3	D_3	PASS
13.2.3	10	PASS, NO_OPTION, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for sched_setparam

Clause	Assertion ID	Conforming Results
13.3.1.1	1	PASS[1, 2], NO_OPTION
13.3.1.1	2	PASS[1, 2], NO_OPTION
13.3.1.1	3	PASS, NO_OPTION
13.3.1.1	4	PASS, NO_OPTION
13.3.1.2	sched_setparam	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.1.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.1.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
13.3.1.2	D_1	PASS, NO_OPTION
13.3.1.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.1.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.1.2	D_2	PASS, NO_OPTION
13.3.1.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.1.2	D_3	PASS, NO_OPTION
13.3.1.2	10	PASS, NO_OPTION, NO_TEST
13.3.1.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
13.3.1.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
13.3.1.2	D_4	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.1.2	D_5	PASS, NO_OPTION
13.3.1.3	R_1	Reference Assertion, no test results
13.3.1.3	R_2	Reference Assertion, no test results
13.3.1.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.1.4	14	PASS, NO_OPTION
13.3.1.4	15	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.1.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.1.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for sched_getparam

Clause	Assertion ID	Conforming Results
13.3.2.1	1	PASS[1, 2], NO_OPTION
13.3.2.1	2	PASS[1, 2], NO_OPTION
13.3.2.1	3	PASS, NO_OPTION
13.3.2.1	4	PASS, NO_OPTION
13.3.2.2	sched_getparam	PASS, NO_OPTION
13.3.2.2	5	PASS, NO_OPTION
13.3.2.2	6	PASS, NO_OPTION
13.3.2.2	D_1	PASS, NO_OPTION
13.3.2.2	D_2	PASS, NO_OPTION
13.3.2.3	R_1	Reference Assertion, no test results
13.3.2.3	R_2	Reference Assertion, no test results
13.3.2.4	7	PASS, NO_OPTION
13.3.2.4	8	PASS, NO_OPTION
13.3.2.4	9	PASS, NO_OPTION

--`,,`-`-`,,`,,`,`,,`---
Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for sched_setscheduler

Clause	Assertion ID	Conforming Results
13.3.3.1	1	PASS[1, 2], NO_OPTION
13.3.3.1	2	PASS[1, 2], NO_OPTION
13.3.3.1	3	PASS, NO_OPTION
13.3.3.1	4	PASS, NO_OPTION
13.3.3.2	sched_setscheduler	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	D_1	PASS, NO_OPTION
13.3.3.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	D_2	PASS, NO_OPTION
13.3.3.2	D_3	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	D_4	PASS, NO_OPTION
13.3.3.2	R_1	Reference Assertion, no test results
13.3.3.2	D_5	PASS, NO_OPTION
13.3.3.3	R_2	Reference Assertion, no test results
13.3.3.3	R_3	Reference Assertion, no test results
13.3.3.4	11	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.4	13	PASS, NO_OPTION
13.3.3.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.4	15	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for sched_getscheduler

Clause	Assertion ID	Conforming Results
13.3.4.1	1	PASS[1, 2], NO_OPTION
13.3.4.1	2	PASS[1, 2], NO_OPTION
13.3.4.1	3	PASS, NO_OPTION
13.3.4.1	4	PASS, NO_OPTION
13.3.4.2	sched_getscheduler	PASS, NO_OPTION
13.3.4.2	5	PASS, NO_OPTION
13.3.4.2	6	PASS, NO_OPTION
13.3.4.2	7	PASS, NO_OPTION
13.3.4.2	8	PASS, NO_OPTION
13.3.4.2	D_1	PASS, NO_OPTION
13.3.4.3	R_1	Reference Assertion, no test results
13.3.4.3	R_2	Reference Assertion, no test results
13.3.4.4	9	PASS, NO_OPTION
13.3.4.4	10	PASS, NO_OPTION
13.3.4.4	11	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Assertions for sched_yield

Clause	Assertion ID	Conforming Results
13.3.5.1	1	PASS[1, 2], NO_OPTION
13.3.5.1	2	PASS[1, 2], NO_OPTION
13.3.5.1	3	PASS, NO_OPTION
13.3.5.1	4	PASS, NO_OPTION
13.3.5.2	sched_yield	PASS, NO_OPTION, NO_TEST
13.3.5.2	D_1	PASS, NO_OPTION
13.3.5.3	R_1	Reference Assertion, no test results
13.3.5.3	R_2	Reference Assertion, no test results
13.3.5.4	5	PASS, NO_OPTION

Assertions for sched_get_priority_max

Clause	Assertion ID	Conforming Results
13.3.6.1	1	PASS[1, 2], NO_OPTION
13.3.6.1	2	PASS[1, 2], NO_OPTION
13.3.6.1	3	PASS, NO_OPTION
13.3.6.1	4	PASS, NO_OPTION
13.3.6.2	sched_get_priority_max	PASS, NO_OPTION
13.3.6.2	14	PASS, NO_OPTION
13.3.6.2	D_1	PASS, NO_OPTION
13.3.6.3	R_1	Reference Assertion, no test results
13.3.6.3	R_3	Reference Assertion, no test results
13.3.6.4	16	PASS, NO_OPTION
13.3.6.4	18	PASS, NO_OPTION
13.3.6.4	19	PASS, NO_OPTION
13.3.6.4	21	PASS, NO_OPTION

Assertions for sched_get_priority_min

Clause	Assertion ID	Conforming Results
13.3.6.1	5	PASS[5, 6], NO_OPTION
13.3.6.1	6	PASS[5, 6], NO_OPTION
13.3.6.1	7	PASS, NO_OPTION
13.3.6.1	8	PASS, NO_OPTION
13.3.6.2	sched_get_priority_min	PASS, NO_OPTION
13.3.6.2	15	PASS, NO_OPTION
13.3.6.2	D_2	PASS, NO_OPTION
13.3.6.3	R_2	Reference Assertion, no test results
13.3.6.3	R_4	Reference Assertion, no test results
13.3.6.4	17	PASS, NO_OPTION
13.3.6.4	22	PASS, NO_OPTION

Assertions for sched_rr_get_interval

Clause	Assertion ID	Conforming Results
13.3.6.1	9	PASS[9, 10], NO_OPTION
13.3.6.1	10	PASS[9, 10], NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.3.6.1	11	PASS, NO_OPTION
13.3.6.1	12	PASS, NO_OPTION
13.3.6.2	sched_rr_get_interval	PASS, NO_OPTION, NO_TEST
13.3.6.2	13	PASS, NO_OPTION, NO_TEST
13.3.6.2	D_3	PASS, NO_OPTION
13.3.6.2	R_5	Reference Assertion, no test results
13.3.6.3	R_6	Reference Assertion, no test results
13.3.6.4	20	PASS, NO_OPTION
13.3.6.4	23	PASS, NO_OPTION

A.14 Clocks and Timers

Assertions for timer_hdr

Clause	Assertion ID	Conforming Results
14.1.1	1	PASS
14.1.1	D_1	PASS, NO_OPTION
14.1.1	2	PASS, NO_TEST
14.1.1	3	PASS
14.1.1	4	PASS, NO_TEST_SUPPORT
14.1.1	5	PASS
14.1.1	6	PASS
14.1.1	D_2	PASS, NO_OPTION
14.1.1	7	PASS, NO_TEST
14.1.1	8	PASS
14.1.1	9	PASS
14.1.1	10	PASS
14.1.1	11	PASS
14.1.2	12	PASS, NO_TEST_SUPPORT
14.1.3	13	PASS
14.1.4	14	PASS
14.1.4	15	PASS
14.1.4	16	PASS
14.1.4	D_3	PASS, NO_OPTION
14.1.4	17	PASS
14.1.4	D_4	PASS, NO_OPTION
14.1.4	D_5	PASS
14.1.4	18	PASS
14.1.4	D_6	PASS

Assertions for clock-settime

Clause	Assertion ID	Conforming Results
14.2.1.1	1	PASS[1, 2], NO_OPTION
14.2.1.1	2	PASS[1, 2], NO_OPTION
14.2.1.1	3	PASS, NO_OPTION
14.2.1.1	4	PASS, NO_OPTION
14.2.1.2	clock_settime	PASS, NO_OPTION, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.2.1.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.1.2	15	PASS
14.2.1.2	17	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.1.2	D_1	PASS
14.2.1.2	18	PASS
14.2.1.2	19	PASS
14.2.1.2	20	PASS
14.2.1.2	21	PASS
14.2.1.2	23	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.1.2	D_2	PASS, NO_OPTION
14.2.1.2	24	PASS, NO_OPTION
14.2.1.2	25	PASS, NO_OPTION
14.2.1.2	D_3	PASS, NO_OPTION
14.2.1.3	R_1	Reference Assertion, no test results
14.2.1.3	R_2	Reference Assertion, no test results
14.2.1.4	26	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.1.4	29	PASS, NO_OPTION
14.2.1.4	32	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.1.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.1.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.1.4	35	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for clock_gettime

Clause	Assertion ID	Conforming Results
14.2.1.1	5	PASS[5, 6], NO_OPTION
14.2.1.1	6	PASS[5, 6], NO_OPTION
14.2.1.1	7	PASS, NO_OPTION
14.2.1.1	8	PASS, NO_OPTION
14.2.1.2	clock_gettime	PASS, NO_OPTION
14.2.1.2	22	PASS, NO_OPTION
14.2.1.2	D_4	PASS, NO_OPTION
14.2.1.3	R_3	Reference Assertion, no test results
14.2.1.3	R_4	Reference Assertion, no test results
14.2.1.4	27	PASS, NO_OPTION
14.2.1.4	30	PASS, NO_OPTION

Assertions for clock_getres

Clause	Assertion ID	Conforming Results
14.2.1.1	9	PASS[9, 10], NO_OPTION
14.2.1.1	10	PASS[9, 10], NO_OPTION
14.2.1.1	11	PASS, NO_OPTION
14.2.1.1	12	PASS, NO_OPTION
14.2.1.1	clock_getres	PASS, NO_OPTION
14.2.1.2	14	PASS, NO_OPTION
14.2.1.2	16	PASS, NO_OPTION
14.2.1.2	D_5	PASS, NO_OPTION
14.2.1.3	R_5	Reference Assertion, no test results

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.2.1.3	R_6	Reference Assertion, no test results
14.2.1.4	28	PASS, NO_OPTION
14.2.1.4	31	PASS, NO_OPTION

Assertions for timer_create

Clause	Assertion ID	Conforming Results
14.2.2.1	1	PASS[1, 2], NO_OPTION
14.2.2.1	2	PASS[1, 2], NO_OPTION
14.2.2.1	3	PASS, NO_OPTION
14.2.2.1	4	PASS, NO_OPTION
14.2.2.2	timer_create	PASS, NO_OPTION
14.2.2.2	5	PASS, NO_OPTION
14.2.2.2	6	PASS
14.2.2.2	7	PASS, NO_OPTION
14.2.2.2	8	PASS, NO_OPTION
14.2.2.2	9	PASS, NO_OPTION
14.2.2.2	10	PASS, NO_OPTION
14.2.2.2	11	PASS, NO_OPTION
14.2.2.2	12	PASS, NO_OPTION
14.2.2.2	13	PASS, NO_OPTION
14.2.2.2	14	PASS, NO_OPTION
14.2.2.2	15	PASS, NO_OPTION
14.2.2.2	D_1	PASS, NO_OPTION
14.2.2.2	16	PASS, NO_OPTION
14.2.2.2	17	PASS, NO_OPTION
14.2.2.2	18	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.2.2	19	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.2.2	D_2	PASS, NO_OPTION
14.2.2.2	20	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.2.2	21	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.2.2	22	PASS, NO_OPTION
14.2.2.2	23	PASS, NO_OPTION
14.2.2.2	24	PASS, NO_OPTION
14.2.2.2	25	PASS, NO_OPTION
14.2.2.3	R_1	Reference Assertion, no test results
14.2.2.3	R_2	Reference Assertion, no test results
14.2.2.3	D_4	PASS, NO_OPTION
14.2.2.4	26	PASS, NO_OPTION, NO_TEST
14.2.2.4	27	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.2.4	28	PASS, NO_OPTION
14.2.2.4	29	PASS, NO_OPTION

Assertions for timer_delete

Clause	Assertion ID	Conforming Results
14.2.3.1	1	PASS[1, 2], NO_OPTION
14.2.3.1	2	PASS[1, 2], NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.2.3.1	3	PASS, NO_OPTION
14.2.3.1	4	PASS, NO_OPTION
14.2.3.2	timer_delete	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.3.2	5	PASS, NO_OPTION
14.2.3.2	D_1	PASS, NO_OPTION
14.2.3.2	D_2	PASS, NO_OPTION
14.2.3.3	R_1	Reference Assertion, no test results
14.2.3.3	R_2	Reference Assertion, no test results
14.2.3.4	6	PASS, NO_OPTION
14.2.3.4	7	PASS, NO_OPTION

Assertions for timer_settime

Clause	Assertion ID	Conforming Results
14.2.4.1	1	PASS[1, 2], NO_OPTION
14.2.4.1	2	PASS[1, 2], NO_OPTION
14.2.4.1	3	PASS, NO_OPTION
14.2.4.1	4	PASS, NO_OPTION
14.2.4.2	timer_settime	PASS, NO_OPTION
14.2.4.2	13	PASS, NO_OPTION
14.2.4.2	14	PASS, NO_OPTION
14.2.4.2	D_1	PASS, NO_OPTION
14.2.4.2	15	PASS, NO_OPTION
14.2.4.2	16	PASS, NO_OPTION
14.2.4.2	17	PASS, NO_OPTION
14.2.4.2	18	PASS, NO_OPTION
14.2.4.2	19	PASS, NO_OPTION
14.2.4.2	20	PASS, NO_OPTION
14.2.4.2	21	PASS, NO_OPTION
14.2.4.2	22	PASS, NO_OPTION
14.2.4.2	23	PASS, NO_OPTION
14.2.4.2	24	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
14.2.4.2	26	PASS, NO_TEST_SUPPORT
14.2.4.2	27	PASS, NO_TEST_SUPPORT
14.2.4.2	D_3	PASS, NO_OPTION
14.2.4.3	R_1	Reference Assertion, no test results
14.2.4.3	R_3	Reference Assertion, no test results
14.2.4.4	32	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.4.4	35	PASS, NO_OPTION
14.2.4.4	38	PASS, NO_OPTION
14.2.4.4	39	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for timer_gettime

Clause	Assertion ID	Conforming Results
14.2.4.1	5	PASS[5, 6], NO_OPTION
14.2.4.1	6	PASS[5, 6], NO_OPTION
14.2.4.1	7	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.2.4.1	8	PASS, NO_OPTION
14.2.4.2	timer_gettime	PASS, NO_OPTION
14.2.4.2	25	PASS, NO_OPTION
14.2.4.2	D_4	PASS, NO_OPTION
14.2.4.3	R_2	Reference Assertion, no test results
14.2.4.3	R_4	Reference Assertion, no test results
14.2.4.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.4.4	36	PASS, NO_OPTION

Assertions for timer_getoverrun

Clause	Assertion ID	Conforming Results
14.2.4.1	9	PASS[9, 10], NO_OPTION
14.2.4.1	10	PASS[9, 10], NO_OPTION
14.2.4.1	11	PASS, NO_OPTION
14.2.4.1	12	PASS, NO_OPTION
14.2.4.2	timer_getoverrun	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
14.2.4.2	28	PASS, NO_OPTION
14.2.4.2	29	PASS, NO_OPTION, NO_TEST
14.2.4.2	30	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
14.2.4.2	31	PASS, NO_OPTION, NO_TEST
14.2.4.2	D_2	PASS, NO_OPTION
14.2.4.2	D_5	PASS, NO_OPTION
14.2.4.3	R_5	Reference Assertion, no test results
14.2.4.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.4.4	37	PASS, NO_OPTION

Assertions for nanosleep

Clause	Assertion ID	Conforming Results
14.2.5.1	1	PASS[1, 2], NO_OPTION
14.2.5.1	2	PASS[1, 2], NO_OPTION
14.2.5.1	3	PASS, NO_OPTION
14.2.5.1	4	PASS, NO_OPTION
14.2.5.2	nanosleep	PASS, NO_OPTION
14.2.5.2	5	PASS, NO_OPTION
14.2.5.2	6	PASS, NO_OPTION, NO_TEST
14.2.5.2	7	PASS, NO_OPTION, NO_TEST
14.2.5.2	8	PASS, NO_OPTION
14.2.5.2	9	PASS, NO_OPTION
14.2.5.2	D_1	PASS, NO_OPTION
14.2.5.3	R_1	Reference Assertion, no test results
14.2.5.3	R_2	Reference Assertion, no test results
14.2.5.3	10	PASS, NO_OPTION
14.2.5.3	11	PASS, NO_OPTION
14.2.5.3	R_3	Reference Assertion, no test results
14.2.5.4	12	PASS, NO_OPTION
14.2.5.4	13	PASS, NO_OPTION
14.2.5.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.2.5.4	15	PASS, NO_OPTION

A.15 Message Passing

Assertions for mq_intro

Clause	Assertion ID	Conforming Results
15.1	D_1	PASS, NO_OPTION

Assertions for mq_hdr

Clause	Assertion ID	Conforming Results
15.1.1	1	PASS
15.1.1	D_1	PASS
15.1.1	2	PASS
15.1.1	3	PASS
15.1.1	D_2	PASS, NO_OPTION
15.1.1	4	PASS, NO_TEST
15.1.1	D_3	PASS, NO_OPTION
15.1.1	GA_mqOpenMaxFD	General Assertion, no test results
15.1.1	GA_mqPCTSOpenMaxFD	General Assertion, no test results

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for mq_open

Clause	Assertion ID	Conforming Results
15.2.1.1	1	PASS[1, 2], NO_OPTION
15.2.1.1	2	PASS[1, 2], NO_OPTION
15.2.1.1	3	PASS, NO_OPTION
15.2.1.1	4	PASS, NO_OPTION
15.2.1.2	OpenMaxMqs	PASS[OpenMaxMqs, PCTSOpenMaxMqs]
15.2.1.2	PCTSOpenMaxMqs	PASS[OpenMaxMqs, PCTSOpenMaxMqs]
15.2.1.2	mq_open	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	D_1	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	5	PASS, NO_OPTION
15.2.1.2	6	PASS, NO_OPTION
15.2.1.2	7	PASS, NO_OPTION
15.2.1.2	8	PASS, NO_OPTION
15.2.1.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	D_2	PASS, NO_OPTION
15.2.1.2	D_3	PASS, NO_OPTION
15.2.1.2	R_1	Reference Assertion, no test results
15.2.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	11	PASS
15.2.1.2	12	PASS
15.2.1.2	13	PASS, NO_OPTION
15.2.1.2	14	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	15	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	16	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	17	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	18	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	19	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	20	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	21	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	22	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	23	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	24	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	D_4	PASS, NO_OPTION
15.2.1.2	D_5	PASS, NO_OPTION
15.2.1.2	25	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	R_2	Reference Assertion, no test results
15.2.1.2	R_3	Reference Assertion, no test results
15.2.1.2	26	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
15.2.1.2	27	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	28	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	D_7	PASS, NO_OPTION
15.2.1.3	R_4	Reference Assertion, no test results
15.2.1.3	R_5	Reference Assertion, no test results
15.2.1.4	mq_open_EACCES1	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	mq_open_EACCES2	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	mq_open_EEXIST	PASS, NO_OPTION, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

15.2.1.4	29	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	30	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	D_8	PASS, NO_OPTION
15.2.1.4	31	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	32	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	35	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	36	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	37	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	mq_open_ENOENT	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	38	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
15.2.1.4	39	PASS, NO_OPTION

Assertions for mq_close

Clause	Assertion ID	Conforming Results
15.2.2.1	1	PASS[1, 2], NO_OPTION
15.2.2.1	2	PASS[1, 2], NO_OPTION
15.2.2.1	3	PASS, NO_OPTION
15.2.2.1	4	PASS, NO_OPTION
15.2.2.2	mq_close	PASS, NO_OPTION
15.2.2.2	D_1	PASS, NO_OPTION
15.2.2.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.2.2	D_2	PASS, NO_OPTION
15.2.2.3	R_1	Reference Assertion, no test results
15.2.2.3	R_2	Reference Assertion, no test results
15.2.2.4	6	PASS, NO_OPTION
15.2.2.4	7	PASS, NO_OPTION

Assertions for mq_unlink

Clause	Assertion ID	Conforming Results
15.2.3.1	1	PASS[1, 2], NO_OPTION
15.2.3.1	2	PASS[1, 2], NO_OPTION
15.2.3.1	3	PASS, NO_OPTION
15.2.3.1	4	PASS, NO_OPTION
15.2.3.2	mq_unlink	PASS, NO_OPTION
15.2.3.2	R_1	Reference Assertion, no test results
15.2.3.2	5	PASS, NO_OPTION
15.2.3.2	D_1	PASS, NO_OPTION
15.2.3.3	R_2	Reference Assertion, no test results
15.2.3.3	R_3	Reference Assertion, no test results
15.2.3.4	6	PASS, NO_OPTION
15.2.3.4	7	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.3.4	mq_unlink_ENOENT	PASS, NO_OPTION
15.2.3.4	8	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for mq_send

Clause	Assertion ID	Conforming Results
15.2.4.1	1	PASS[1, 2], NO_OPTION
15.2.4.1	2	PASS[1, 2], NO_OPTION
15.2.4.1	3	PASS, NO_OPTION
15.2.4.1	4	PASS, NO_OPTION
15.2.4.2	mq_send	PASS, NO_OPTION
15.2.4.2	R_1	Reference Assertion, no test results
15.2.4.2	5	PASS, NO_OPTION
15.2.4.2	6	PASS, NO_OPTION
15.2.4.2	7	PASS, NO_OPTION
15.2.4.2	R_2	Reference Assertion, no test results
15.2.4.2	8	PASS, NO_OPTION
15.2.4.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.4.2	D_1	PASS, NO_OPTION
15.2.4.3	R_4	Reference Assertion, no test results
15.2.4.3	R_5	Reference Assertion, no test results
15.2.4.4	mq_send_EAGAIN	PASS, NO_OPTION
15.2.4.4	10	PASS, NO_OPTION
15.2.4.4	11	PASS, NO_OPTION
15.2.4.4	mq_send_EINVAL	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.4.4	mq_send_EMSGSIZE	PASS, NO_OPTION
15.2.4.4	12	PASS, NO_OPTION

Assertions for mq_receive

Clause	Assertion ID	Conforming Results
15.2.5.1	1	PASS[1, 2], NO_OPTION
15.2.5.1	2	PASS[1, 2], NO_OPTION
12.2.5.1	3	PASS, NO_OPTION
12.2.5.1	4	PASS, NO_OPTION
12.2.5.2	mq_receive	PASS, NO_OPTION
12.2.5.2	5	PASS, NO_OPTION
12.2.5.2	6	PASS, NO_OPTION
12.2.5.2	7	PASS, NO_OPTION
12.2.5.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.5.2	D_1	PASS, NO_OPTION
12.2.5.2	R_1	Reference Assertion, no test results
12.2.5.2	D_2	PASS, NO_OPTION
12.2.5.3	R_2	Reference Assertion, no test results
12.2.5.3	R_3	Reference Assertion, no test results
12.2.5.4	mq_receive_EAGAIN	PASS, NO_OPTION
12.2.5.4	9	PASS, NO_OPTION
12.2.5.4	10	PASS, NO_OPTION
12.2.5.4	11	PASS, NO_OPTION
12.2.5.4	12	PASS, NO_OPTION
12.2.5.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Assertions for mq_notify

Clause	Assertion ID	Conforming Results
15.2.6.1	1	PASS[1, 2], NO_OPTION
15.2.6.1	2	PASS[1, 2], NO_OPTION
15.2.6.1	3	PASS, NO_OPTION
15.2.6.1	4	PASS, NO_OPTION
15.2.6.2	mq_notify	PASS, NO_OPTION
15.2.6.2	R_1	Reference Assertion, no test results
15.2.6.2	5	PASS, NO_OPTION
15.2.6.2	6	PASS, NO_OPTION
15.2.6.2	7	PASS, NO_OPTION
15.2.6.2	D_1	PASS, NO_OPTION
15.2.6.3	R_2	Reference Assertion, no test results
15.2.6.3	R_3	Reference Assertion, no test results
152.2.6.4	8	PASS, NO_OPTION
15.2.6.4	mq_notify_EBUSY	PASS, NO_OPTION
15.2.6.4	9	PASS, NO_OPTION

Assertions for mq_setattr

Clause	Assertion ID	Conforming Results
15.2.7.1	1	PASS[1, 2], NO_OPTION
15.2.7.1	2	PASS[1, 2], NO_OPTION
15.2.7.1	3	PASS, NO_OPTION
15.2.7.1	4	PASS, NO_OPTION
15.2.7.2	mq_setattr	PASS, NO_OPTION
15.2.7.2	5	PASS, NO_OPTION
15.2.7.2	D_1	PASS, NO_OPTION
15.2.7.2	6	PASS, NO_OPTION
15.2.7.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.7.2	D_2	PASS, NO_OPTION
15.2.7.3	R_1	Reference Assertion, no test results
15.2.7.3	R_2	Reference Assertion, no test results
15.2.7.4	8	PASS, NO_OPTION
15.2.7.4	9	PASS, NO_OPTION

Assertions for mq_getattr

Clause	Assertion ID	Conforming Results
15.2.8.1	1	PASS[1, 2], NO_OPTION
15.2.8.1	2	PASS[1, 2], NO_OPTION
15.2.8.1	3	PASS, NO_OPTION
15.2.8.1	4	PASS, NO_OPTION
15.2.8.2	mq_getattr	PASS, NO_OPTION
15.2.8.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.8.2	6	PASS, NO_OPTION
15.2.8.2	7	PASS, NO_OPTION
15.2.8.2	D_1	PASS, NO_OPTION
15.2.8.3	R_1	Reference Assertion, no test results

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

15.2.8.3	R_2	Reference Assertion, no test results
15.2.8.3	8	PASS, NO_OPTION
15.2.8.4	9	PASS, NO_OPTION
15.2.8.4	10	PASS, NO_OPTION

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex B
(informative)

Bibliography

This Annex contains lists of related open systems standards and suggested reading on historical implementations and application programming.

B.1 Related Open Systems Standards

B.1.1. Networking Standards

{B1} ISO 7498:1984, *Information processing systems—Open Systems Interconnection—Basic Reference Model.* [4)]

{B2} ISO 8072:1986, *Information processing systems—Open Systems Interconnection—Transport service definition.*

{B3} ISO/IEC 8073:1988, *Information processing systems—Open Systems Interconnection—Connection oriented transport protocol specification.* [5)]

{B4} ISO 8326:1987, *Information processing systems—Open Systems Interconnection—Basic connection oriented session service definition.*

{B5} ISO 8327:1987, *Information processing systems—Open Systems Interconnection—Basic connection oriented session protocol definition.*

{B6} ISO 8348:1987, *Information processing systems—Data communications—Network service definition.*

{B7} ISO 8473:1988, *Information processing systems—Data communications—Protocol for providing the connectionless-mode network service.*

{B8} ISO 8571:1988, *Information processing systems—Open Systems Interconnection—File Transfer, Access and Management.*

---

4) ISO/IEC documents are available from the ISO Central Secretariat, Case Postal 56, 1 rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/). ISO/IEC publications are available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112, USA (http://global.ihs.com/). Electronic copies are available in the United States from the American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA (http://www.ansi.org/).

5) IEC documents can be obtained from the IEC office, 3, rue de Varembé, Case Postale 131, CH-1211, Genève 20, Switzerland/Suisse.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

{B9} ISO 8649:1988, *Information processing systems—Open Systems Interconnection—Service definition for the Association Control Service Element.*

{B10} ISO 8650:1988, *Information processing systems—Open Systems Interconnection—Protocol specification for the Association Control Service Element.*

{B11} ISO 8802-2:1989 [IEEE Std 802.2-1989 (ANSI)], *Information processing systems—Local area networks—Part 2: Logical link control.*

{B12} ISO 8802-3:1989 [IEEE 802.3-1988 (ANSI)], *Information processing systems—Local area networks—Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications.*

{B13} ISO/IEC 8802-4:1990 [IEEE Std 802.4-1990 (ANSI)], *Information technology—Local area networks—Part 4: Token-passing bus access method and physical layer specifications.*

{B14} ISO 8802_5: ... (IEEE 802.5-1989), *Information technology—Local area networks—Part 5: Token ring access method and physical layer specifications.*

{B15} ISO 8822:1988, *Information processing systems—Open Systems Interconnection—Connection oriented presentation service definition.*

{B16} ISO 8823:1988, *Information processing systems—Open Systems Interconnection—Connection oriented presentation protocol specification.*

{B17} ISO 8831:1989, *Information processing systems—Open Systems Interconnection—Job transfer and manipulation concepts and services.*

{B18} ISO 8832:1989, *Information processing systems—Open Systems Interconnection—Specification of the basic class protocol for job transfer and manipulation.*

{B19} CCITT Recommendation X.25, *Interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCT) for terminals operating in the packet mode and connected to public data networks by dedicated circuit.*[6)]

{B20} CCITT Recommendation X;.212, *Information processing systems—Data communication—Data link service definition for Open Systems Interconnection.*

B.1.2 Language Standards

{B21} ISO 1539:1980, *Programming languages—FORTRAN.*

{B22} ISO 1989:1985, *Programming Languages—COBOL.*

{B23} ISO 8652:1987, *Programming Languages—Ada.*

{B24} ANSI X3.113_1987 [7)], *Information systems—Programming language—FULL BASIC.*

{B25} ANSI/IEEE 770X3.97-1983, *Standard Pascal Computer Programming Language.*

{B26} ANSI/MDC X11.1-1984, *Programming Language MUMPS.*

---

6) CCITT documents can be obtained from the CCITT General Secretariat, International Telecommunications Union, Sales Section, Place des Nations, CH-1211, Genève 20, Switzerland/Suisse.

7) ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY 10018.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

B.1.3 Graphics Standards

{B27} ISO 7942:1985, *Information processing systems—Computer graphics—Graphical Kernel System (GKS) functional description.*

{B28} ISO 8632:1987, *Information processing systems—Computer graphics—Metafile for the storage and transfer of picture description information.*

{B29} ISO/IEC 9592:1989 (ANSI X3.144-1988), *Information processing systems—Computer graphics—Programmer's hierarchical interactive graphics system (PHIGS).*

B.1.4 Database Standards

{B30} ISO 8907:1987, *Database Language—NDL.*

{B31} ISO 9075:1987, *Database Language—SQL.*

B.2 Other Standards

{B32} ISO 639:1988, *Code for the representation of names of languages.*

{B33} ISO 3166:1988, *Code for the representation of names of countries.*

{B34} ISO 8859-1:1987, *Information Processing—8-bit single-byte coded graphic character sets—Part 1: Latin alphabet No. 1.*

{B35} ISO 9127:1988, *Information processing systems—User documentation and cover information for consumer software packages.*

{B36} ISO/IEC 9945-2: ...,[8] *Information technology—Portable operating system interface (POSIX)—Part 2: Shell and utilities.*

{B37} ISO/IEC 10646:...,[9] *Information processing—Multiple octet coded character set.*

{B38} IEEE Std 100-1988, *IEEE Standard Dictionary of Electrical and Electronics Terms.*

{B39} P1003.0/D16,[10] *Draft Guide to the POSIX Open Systems Environment.*

{B40} ISO/IEC TR 10000-1:1990, *Information technology—Framework and taxonomy of International Standardized Profiles—Part 1: Framework.*

B.3 Historical Documentation and Introductory Texts

{B41} American Telephone and Telegraph Company, *System V Interface Definition (SVID), Issues 2 and 3.* Morristown, NJ: UNIX Press, 1986, 1989.[11]

---

8) To be approved and published.

9) To be approved and published.

10) This unapproved draft document is available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (http://standards.ieee.org/).

11) This is one of several documents that represent an industry specification in an area related to POSIX.1. The creators of such documents may be able to identify newer versions that may be of interest to the reader.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

{B42} American Telephone and Telegraph Company. *UNIX System III Programmer's Manual.* Greensboro, NC: Western Electric Company, October 1981.

{B43} American Telephone and Telegraph Company. *UNIX Time Sharing System: UNIX Programmer's Manual.* 7th ed. Murray Hill, NJ: Bell Laboratories, January 1979.

{B44} "The UNIX System."[12] *AT&T Bell Laboratories Technical Journal.* vol. 63 (8 Part 2), October 1984.

{B45} "UNIX Time-Sharing System."[13] *Bell System Technical Journal.* vol. 57 (6 Part 2), July-August 1978.

{B46} Bach, Maurice J. *The Design of the UNIX Operating System.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

{B47} Dijkstra, E.W. "Solution of a Problem in Concurrent Programming Control," *Communications of the ACM,* vol. 8(9), September 1965, pp. 569-570.

{B48} Furht, Borko, Grostick, Dan, Gluch, David, Rabbat, Guy, Parker, John, and McRoberts, Meg. *Real-Time UNIX Systems: Design and Application Guide.* Boston, MA: Kluwer Academic Publishers, 1991.

{B49} Harbison, Samuel P. and Steele, Guy L. *C: A Reference Manual.* Englewood Cliffs, NJ: Prentice_Hall, 1987.

{B50} Kernighan, Brian W. and Ritchie, Dennis M. *The C Programming Language.* Englewood Cliffs, NJ: Prentice-Hall, 1978.

{B51} Kernighan, Brian W. and Pike, Rob. *The UNIX Programming Environment.* Englewood Cliffs, NJ: Prentice-Hall, 1984.

{B52} Leffler, Samuel J., McKusick, Marshall Kirk, Karels, Michael J. , Quarterman, John S., and Stettner, Armando. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Reading, MA: Addison-Wesley, 1988.

{B53} McGilton, Henry and Morgan, Rachel. *Introducing the UNIX System.* New York: McGraw-Hill (BYTE BOOKS), 1983.

{B54} Organick, Elliot I. *The Multics System: An Examination of Its Structure.* Cambridge, MA: the MIT Press, 1972.

{B55} Quarterman, John S., Silberschatz, Abraham, and Peterson, James L. "4.2BSD and 4.3BSD as Examples of the UNIX System." *ACM Computing Surveys.* vol. 17(4), December 1985, pp. 379-418.

{B56} Ritchie, Dennis M. "Reflections on Software Research." *Communiciations of the ACM, vol. 27(8),* August 1984, pp. 758-760. ACM Turing Award Lecture.

{B57} Ritchie, Dennis. "The Evolution of the UNIX Time-Sharing System." *AT&T Bell Laboratories Technical Journal.* vol. 63(8), October 1984, pp. 1577-1593.

{B58} Ritchie, D.M. and Thompson, K. "The UNIX Time-Sharing System." *Communications of the ACM.* vol.7(7), July 1974, pp. 365-375. This is the original paper, which describes Version 6.

{B59} Ritchie, D.M. and Thompson, K. "The UNIX Time-Sharing System." *Bell System Technical Journal.* vol. 57 (6 Part 2), July-August 1978, pp. 1905-1929. This is a revised version and describes Version 7.

12) This entire edition is devoted to the UNIX system.

13) This entire edition is devoted to the UNIX time-sharing system.

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

{B60} Ritchie, Dennis M. "Unix: A Dialectic." *Winter 1987 USENIX Association Conference Proceedings, Washington, D.C.,* pp. 29-34. Berkeley, CA: USENIX Association, January 1987.

{B61} Rochkind, Marc J. *Advanced UNIX Programming.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

{B62} University of California at Berkeley—Computer Science Research Group. *4.3 Berkeley Software Distribution, Virtual VAX-11 Version.* Berkeley, CA: The Regents of the University of California, April 1986.

{B63} /usr/group Standards Committee. *1984 /usr/group Standard.* Santa Clara, CA: Uni_Forum, 1984.

{B64} X/Open Company, Ltd. *X/Open Portability Guide, Issue 2.* Amsterdam: Elsevier Science Publishers, 1987.

{B65} X/Open Company, Ltd. *X/Open Portability Guide, Issue 3.* Englewood Cliffs, NJ: Prentice-Hall, 1989.

B.4 Other Sources of Information

{B66} ISO/IEC JTC 1 N1335, *Final Report of ISO/IEC JTC 1 TSG-1 on Standards Necessary to Define Interfaces for Application Portability (IAP).*

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Identifier Index

access() Check File Accessibility {5.6.3} . . . 89
aio_cancel() Cancel Asynchronous I/O Request {6.7.7} . . . 141
aiocb() Asynchronous I/O Control Block {6.7.1.1} . . . 114
aio_error() Retrieve Error Status of Asynchronous I/O Operation {6.7.5} . . . 138
aio_fsync() Asynchronous File Synchronization {6.7.9} . . . 147
<aio.h> Asynchronous I/O Control Block {6.7.1.1} . . . 112
aio_read() Asynchronous Read {6.7.2} . . . 115
aio_return() Retrieve Return Status of Asynchronous I/O Operation {6.7.6} . . . 140
aio_suspend() Wait for Asynchronous I/O Request {6.7.8} . . . 144
aio_write() Asynchronous Write {6.7.3} . . . 121
alarm() Schedule Alarm {3.4.1} . . . 76
chdir() Change Current Working Directory {5.2.1} . . . 82
chmod() Change File Modes {5.6.4} . . . 89
chown() Change Owner and Group of a File {5.6.5} . . . 92
clock_getres() Clocks {14.2.1} . . . 247
clock_gettime() Clocks {14.2.1} . . . 247
clockid_t Type Definitions {14.1.3} . . . 246
clock_settime() Clocks {14.2.1} . . . 247
close() Close a File {6.3.1} . . . 100
closedir() Director Operations {5.1.2} . . . 82
creat() Create a New File or Rewrite an Existing One {5.3.2} . . . 84
dev_t Primitive System Data Types {2.5} . . . 35
directory Directory Operations {5.1.2} . . . 82
dup() Duplicate an Open File Descriptor {6.2.1} . . . 99
dup2() Duplicate an Open File Descriptor {6.2.1} . . . 99
<errno.h> Error Numbers {2.4} . . . 34
execl() Execute a File {3.1.2} . . . 50
execle() Execute a File {3.1.2} . . . 50
execlp() Execute a File {3.1.2} . . . 50
execv() Execute a File {3.1.2} . . . 50
execve() Execute a File {3.1.2} . . . 50
execvp() Execute a File {3.1.2} . . . 50
_exit() Terminate a Process {3.2.2} . . . 53
fchmod() Change File Modes {5.6.4} . . . 89
fcntl() File Control {6.5.2} . . . 103
<fcntl.h> Data Definitions for File Control Operations {6.5.1} . . . 103
fdatasync() Synchronize the Data of a File {6.6.2} . . . 110
fdopen() Open a Stream on a File Descriptor {8.2.2} . . . 154
fork() Process Creation {3.1.1} . . . 48
fpathconf() Get Configurable Pathname Variables {5.7.1} . . . 96
fstat() Get File Status {5.6.2} . . . 88
fsync() Synchronize the State of a File {6.6.1} . . . 107
ftruncate() Truncate a File to a Specified Length {5.6.7} . . . 93
getcwd() Get Working Directory Pathname {5.2.2} . . . 83
gid_t Primitive System Data Types {2.5} . . . 35
ino_t Primitive System Data Types {2.5} . . . 35
kill() Send a Signal to a Process {3.3.2} . . . 65
<limits.h> Minimum Values {2.8.2} . . . 40
link() Link to a File {5.3.4} . . . 85
lio_listio() List Directed I/O {6.7.4} . . . 128
lseek() Reposition Read/Write File Offset {6.5.3} . . . 106
mkdir() Make a Directory {5.4.1} . . . 85
mkfifo() Make a FIFO Special File {5.4.2} . . . 85
mlock() Lock/Unlock a Range of Process Address Space {12.1.2} . . . 193
mlockall() Lock/Unlock the Address Space of a Process {12.1.1} . . . 189
mmap() Map Process Addresses to a Memory Object {12.2.1} . . . 197
mode_t Primitive System Data Types {2.5} . . . 35

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

mprotect() Change Memory Protection {12.2.3} . . . 207
mq_attr() Data Structures {15.1.1} . . . 269
mq_close() Close a Message Queue {15.2.2} . . . 278
mqd_t() Data Structures {15.1.1} . . . 269
mq_getattr() Get Message Queue Attributes {15.2.8} . . . 291
mq_notify() Notify Process that a Message is Available on a Queue {15.2.6} . . . 287
mq_open() Open a Message Queue {15.2.1} . . . 270
mq_receive() Receive a Message from a Message Queue {15.2.5} . . . 285
mq_send() Send a Message to a Message Queue {15.2.4} . . . 282
mq_setattr() Set Message Queue Attributes {15.2.7} . . . 289
<mqueue.h> Data Structures {15.1.1} . . . 269
mq_unlink() Remove a Message Queue {15.2.3} . . . 280
msync() Memory Object Synchronization {12.2.4} . . . 211
munlock() Lock/Unlock a Range of Process Address Space {12.1.2} . . . 193
munlockall() Lock/Unlock the Address Space of a Process {12.1.1} . . . 189
munmap() Unmap Previously Mapped Addresses {12.2.2} . . . 205
nanosleep() High Resolution Sleep {14.2.5} . . . 265
nlink_t() Primitive System Data Types {2.5} . . . 35
off_t() Primitive System Data Types {2.5} . . . 35
open() Open a File {5.3.1} . . . 83
opendir() Directory Operations {5.1.2} . . . 82
pathconf() Get Configurable Pathname Variables {5.7.1} . . . 96
pause() Suspend Process Execution {3.4.2} . . . 76
pid_t() Primitive System Data Types {2.5} . . . 35
pipe() Create an Inter-Process Channel {6.1.1} . . . 99
read() Read from a File {6.4.1} . . . 101
readdir() Directory Operations {5.1.2} . . . 82
rename() Rename a File {5.5.3} . . . 86
rewinddir() Directory Operations {5.1.2} . . . 82
rmdir() Remove a Directory {5.5.2} . . . 86
sched_getparam() Get Scheduling Parameters {13.3.2} . . . 232
sched_get_priority_max()
Get Scheduling Parameter Limits {13.3.6} . . . 240
sched_get_priority-min()
Get Scheduling Parameter Limits {13.3.6} . . . 240
sched_getscheduler()
Get Scheduling Policy {13.3.4} . . . 238
<sched.h> Scheduling Parameters {13.1} . . . 225
sched_param Scheduling Parameters {13.1} . . . 225
sched_rr_get_interval()
Get Scheduling Parameter Limits {13.3.6} . . . 240
sched_setparam() Set Scheduling Parameters {13.3.1} . . . 229
sched_setscheduler()
Set Scheduling Policy and Scheduling Parameters {13.3.3} . . . 234
sched_yield() Yield Processor {13.3.5} . . . 239
<semaphore.h> Semaphore Characteristics {11.1} . . . 157
sem_close() Close a Named Semaphore {11.2.4} . . . 168
sem_destroy() Destroy an Unnamed Sempahore {11.2.2} . . . 161
sem_getvalue() Get the Value of a Sempahore {11.2.8} . . . 184
sem_init() Initialize an Unnamed Semaphore {11.2.1} . . . 158
sem_open() Initialize/Open a Named Semaphore {11.2.3} . . . 162
sem_post() Unlock a Semaphore {11.2.7} . . . 180
sem_t() Semaphore Characteristics {11.1} . . . 157
sem_trywait() Lock a Semaphore {11.2.6} . . . 172
sem_unlink() Remove a Named Sempahore {11.2.5} . . . 170
sem_wait() Lock a Semaphore {11.2.6} . . . 172
shm_open() Open a Shared Memory Object {12.3.2} . . . 215
shm_unlink() Remove a Shared Memory Object {12.3.2} . . . 222
sigaction() Examine and Change Signal Action {3.3.4} . . . 65
sigaddset() Manipulate Signal Sets {3.3.3} . . . 65

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

sigdelset() Manipulate Signal Sets {3.3.3} . . . 65
sigemptyset() Manipulate Signal Sets {3.3.3} . . . 65
sigevent() Signal Generation and Delivery {3.3.1.2} . . . 57
sigfillset() Manipulate Signal Sets {3.3.3} . . . 65
siginfo_t() Signal Actions {3.3.1.3} . . . 61
siginfo_t() Signal Actions {3.3.1.3} . . . 61
siginfo_t() Signal Actions {3.3.1.3} . . . 61
siginfo_t() Signal Actions {3.3.1.3} . . . 61
siginfo_t() Signal Actions {3.3.1.3} . . . 61
siginfo_t() Signal Actions {3.3.1.3} . . . 61
siginfo_t() Signal Actions {3.3.1.3} . . . 61
siginfo_t() Signal Actions {3.3.1.3} . . . 61
siginfo_t() Signal Actions {3.3.1.3} . . . 62
siginfo_t() Signal Actions {3.3.1.3} . . . 62
siginfo_t() Signal Actions {3.3.1.3} . . . 62
siginfo_t() Signal Actions {3.3.1.3} . . . 66
siginfo_t() Signal Actions {3.3.1.3} . . . 62
sigismember() Manipulate Signal Sets {3.3.3} . . . 65
sigpending() Examine Pending Signals {3.3.6} . . . 67
sigprocmask() Examine and Change Blocked Signals {3.3.5} . . . 67
sigqueue() Queue a Signal to a Process {3.3.9} . . . 72
sigsetops() Manipulate Signal Sets {3.3.3} . . . 65
sigsuspend() Wait for a Signal {3.3.7} . . . 68
sigtimedwait() Synchronously Accept a Signal {3.3.8} . . . 68
sigwaitinfo() Synchronously Accept a Signal {3.3.8} . . . 68
size_t() Primitive System Data Types {2.5} . . . 35
sleep() Delay Process Execution {3.4.3} . . . 76
ssize_t() Primitive System Data Types {2.5} . . . 35
stat() Get File Status {5.6.2} . . . 88
sysconf() Get Configurable System Variables {4.8.1} . . . 78
`<sys/stat.h>` File Characteristics: Header and Data Structure {5.6.1} . . . 87
`<sys/types.h>` Primitive System Data Types {2.5} . . . 35
`<sys/wait.h>` Wait for Process Termination {3.2.1} . . . 53
`<time.h>` Time Value Specification Structures {14.1.1} . . . 245
`<time.h>` Time Value Specification Structures {14.1.1} . . . 245
timer_create() Create a Per-Process Timer {14.2.2} . . . 253
timer_delete() Delete a Per-Process Timer {14.2.3} . . . 257
timer_getoverrun() Per-Process Timers {14.2.4} . . . 259
timer_gettime() Per-Process Timers {14.2.4} . . . 259
timer_settime() Per-Process Timers {14.2.4} . . . 259
timer_t Type Definitions {14.1.3} . . . 246
time_t Primitive System Data Types {2.5} . . . 35
uid_t Primitive System Data Types {2.5} . . . 35
umask() Set File Creation Mask {5.3.3} . . . 84
unlink() Remove Directory Entries {5.5.1} . . . 86
utime() Set File Creation Mask {5.5.3} . . . 93
wait Wait for Process Termination {3.2.1} . . . 53
waitpid() Wait for Process Termination {3.2.1} . . . 53
write() Write to a File {6.4.2} . . . 102

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Alphabetic Topical Index

mq_close()
function definition 278
pathconf()
function definition 96
rename()
function definition 86
sched_getparam()
function definition 232
(time) resolution
definition of 24
<aio.h>
header definition 112
<fcntl.h>
header definition 103
<mqueue.h>
header definition 269
<sched.h>
header definition 225
<semaphore.h>
header definition 157
<sys/stat.h>
File Modes 88
File Types 87
header definition 87
Time Entries 88
<time.h>
header definition 245
_exit
function definition 53
GAP 6
RAP 6
_sc_aio_listio_max
limit definition 80
_sc_aio_max
limit definition 80
_sc_aio_prio_delta_max
limit definition 80
_sc_asynchronous_io
limit definition 80
_sc_delaytimer_max
limit definition 80
_sc_fsync
limit definition 80
_sc_mapped_files
limit definition 80
_sc_memlock
limit definition 80
_sc_memlock_range
limit definition 80
_sc_memory_protection
limit definition 80
_sc_message_passing
limit definition 80
_sc_mq_open_max
limit definition 80
_sc_mq_prio_max
limit definition 80
_sc_pagesize
limit definition 80
_sc_prioritized_io
limit definition 80
_sc_priority_scheduling
limit definition 80
_sc_realtime_signals
limit definition 80
_sc_sem_nsems_max
limit definition 80
_sc_sem_value_max
limit definition 80
_sc_semaphores
limit definition 80
_sc_shared_memory_objects
limit definition 80
_sc_sigqueue_max
limit definition 80
_sc_synchronized_io
limit definition 80
_sc_timer_max
limit definition 80
_sc_timers
limit definition 80
Abbreviations 5
absolute pathname
definition of 20
access mode
definition of 20
access()
function definition 89
ACM 344
address space
definition of 20
aio_alldone
definition of 115
aio_cancel()
function definition 141
aio_canceled 115
aio_error()
function definition 138
aio_fsync()
function definition 147

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

aio_listio_max 78, 80
aio_max ... 79, 80, 118, 119, 124, 132, 151
aio_notcanceled
definition of 115
aio_prio_delta_max 79, 80
aio_read()
function definition 115
aio_return()
function definition 140
aio_suspend()
function definition 144
aio_write()
function definition 121
aiocb
C-language type definition 112
alarm
schedule 76
alarm()
function definition 76
ANSI 2, 341-343
ANSI/IEEE 342
ANSI/MDC 342
appropriate privileges . 7, 10, 20, 28, 47, 91, 92, 161, 197, 236
arm (a timer)
definition of 20
Asynchronous File Synchronization 147
asynchronous I/O completion
definition of 20
Asynchronous I/O Control Block 112
asynchronous I/O operation
definition of 20
Asynchronous Input and Output 112
Asynchronous Read 115
Asynchronous Write 121
AT&T 344
background 20, 56, 126, 133, 134, 136
background process
definition of 20
background process group
definition of 20
BASIC 342
Bibliography ..... viii, 341
blocked process
definition of 20
BSD
4.2 344
4.3 344
BYTE 343, 344
C language
Language -Dependent Services 3
C Standard
Abbreviation 5
definition of 5
C Standard Language-Dependent System Support 3
C-language type definitions
aiocb 112
clockid_t 246
mq_attr 269
mqd_t 269
sched_param 225
sem_t 157
sigevent 57
siginfo_t 57, 61, 62, 66, 68
timer_t 246
Cancel Asynchronous i/o Request 141
CCITT 342
Change Current Working Directory 82
Change File Modes 89
Change Memory Protection 207
Change Owner and Group of a File 92
character
definition of 20
character special file
definition of 20
chdir()
function definition 82
Check File Accessibility 89
child process
definition of 20
chmod()
function definition 89
chown()
function definition 92
CH-1211 ..... 2, 341, 342
clk_tck 81
definition of 81
clock
definition of 20
clock tick 80
definition of 20
clock_getres
Reference Assertion definition 249
clock_getres()
function definition 247
clock_gettime
Reference Assertion definition 249
clock_gettime()
function definition 247
clock_realtime 266
definition of 247
clock_settime
Reference Assertion definition 248
clock_settime()
function definition 247
clockid_t
C-language type definition 246
Clocks 247

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Clocks and Timer Functions 247
Clocks and Timers 245, 330
definition of 245
Close a File 100
Close a Message Queue 278
Close a Named Semaphore 168
close()
function definition 100
closedir()
function definition 82
Common-Usage C Language-Dependent System Support 3
Common-Usage C Language-Dependent System Support 3
Configurable Pathname Variables 96
Configurable System Variables .. 78, 80, 349
conformance . 1, viii, 1-5, 21, 22, 24-40, 42-46, 48-64, 66-75, 78, 79, 81, 84, 87-97, 100-152, 154, 157-223, 225-267, 269-293
definition of 3
conformance requirements
definition of 3
conforming implementation
definition of 3
Conforming Implementation Options 3
Conforming Test Results 293
constant 78, 79
Control Operations on Files 103
controlling process
definition of 20
controlling terminal
definition of 20
Conventions 4, 19
creat()
function definition 84
Create a Per-Process Timer 253
Create an Inter-Process Channel 99
Create New File or Rewrite an Existing One 84
Cross References 5
CSMA/CD 342
current working directory
change 82
definition of 20
D.C. 344
D_1 ... 24, 38, 44, 51, 55, 56, 66, 67, 69, 74, 87, 94, 97, 117, 122, 129, 140, 143, 146, 150, 154, 159, 161, 164, 169, 171, 175, 182, 185, 187, 190, 194, 198, 206, 208, 212, 216, 223, 225, 226, 230, 233, 235, 239, 240, 243, 245, 249, 250, 255, 258, 260, 266, 269, 271, 279, 281, 283, 286, 288, 290, 292, 293, 295-298, 300, 301, 304-321, 323, 324, 326-339
D_10 ..... 167, 221, 316, 325
D_2 . 24, 40, 43, 51, 55, 57, 70, 94, 113, 117, 123, 141, 143, 159, 162, 165, 169, 175, 183, 187, 191, 195, 200, 206, 209, 212, 216, 228, 230, 233, 235, 243, 246, 249, 255, 258, 263, 269, 272, 279, 283, 286, 290, 293, 295-298, 300, 301, 305, 308-310, 312, 313, 315-321, 323, 324, 326-339
D_3 . 27, 52, 57, 70, 113, 117, 123, 159, 162, 165, 169, 187, 191, 195, 200, 206, 209, 212, 216, 228, 230, 236, 243, 247, 250, 256, 263, 270, 272, 293, 296-298, 301, 308-310, 315-317, 319, 320, 322-324, 326-328, 330, 331, 333, 335, 336
D_4 .... 53, 58, 71, 114, 118, 124, 162, 165, 188, 191, 195, 200, 209, 212, 217, 231, 236, 247, 250, 257, 263, 274, 296-298, 300, 301, 309, 310, 315, 316, 319, 320, 322-325, 327, 328, 330-332, 334, 336
D_5 ... 59, 71, 165, 189, 192, 195, 201, 212, 218, 231, 236, 247, 250, 263, 274, 298, 301, 316, 319, 320, 322, 324, 325, 327, 328, 330, 331, 334, 336
D_6 .. 61, 166, 193, 196, 201, 213, 219, 275, 299, 316, 320-322, 324, 325, 330
D_7 .. 62, 166, 197, 202, 213, 219, 275, 299, 316, 321, 322, 324, 325, 336
D_8 .. 63, 166, 197, 202, 220, 277, 299, 316, 321, 322, 325, 337
D_9 ..... 63, 166, 220, 299, 316, 325
Data Definitions for Asynchronous Input and Output 112
Data Definitions for Clocks and Timers . 245
Data Definitions for File Control Operations 103
Data Definitions for Message Queues .. 269
Data Interchange Format 156, 314
definition of 156
Data Structures 269
Database Standards 343
DCT 342
Definitions 20
Delay Process Execution 76
delaytimer_max 78, 262
Delete a Per-Process Timer 257
Destroy an Unnamed Semaphore 161
device
definition of 20
Device- and Class- Specific Functions .. 153

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

definition of 153
Device- and Class-Specific Functions .. 314
DGA01 3
direct I/O
definition of 20
Directories 82
directory .. 20, 21, 23, 24, 27, 30-34, 36, 82, 83, 85, 86, 96, 97, 303, 347-349
change current working 82
current working 20
definition of 20
empty 21
make 85
parent 23
remove 86
root 24
directory entry
definition of 20
remove 86
directory entry [link]
definition of 20
Directory Operations 82
disarm (a timer)
definition of 20
document ..... 1, i, viii, ix, 4, 5, 38, 343
Documentation 3
dot
definition of 21
dot-dot
definition of 21
drift rate (of a clock)
definition of 21
DTE 342
dup()
function definition 99
dup2()
function definition 99
Duplicate an Open File Descriptor 99
effective group 274
effective group id 21, 47, 91, 164, 274
definition of 21
effective user id .. 8, 21, 72, 90-92, 164, 218, 274
definition of 21
empty directory
definition of 21
empty string
definition of 21
Epoch
definition of 21
Examine and Change Blocked Signals ... 67
Examine and Change Signal Action 65
Examine Pending Signals 67
execl()
function definition 50
execle()
function definition 50
execlp()
function definition 50
Execute a File 50
Execution Scheduling 225, 326
definition of 225
execv()
function definition 50
execve()
function definition 50
f_dupd 106
f_dupfd 104, 105
f_getfd 104
f_getfl 104
f_setfd 104
f_setfl 105
fchmod ()
function definition 89
fcntl()
function definition 103
fdatasync()
function definition 110
fdopen()
function definition 154
feature test macro
definition of 21
fifo ... 21, 49, 51, 58, 66, 85, 113, 120, 125-127, 134, 136, 137, 182, 226-228, 231, 242, 347
make 85
fifo scheduling policy
definition of 226
FIFO special file
definition of 21
file
access 89
block special 20
change modes 89
change owner and group 92
character special 20
close 100
create or rewrite 84
definition of 21
execute 50
FIFO special 21
link to 85
make FIFO special 85
open 83
read 101
regular 24
rename 86
write 102
File Characteristics 87

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

File Characteristics: Header and Data Structure 86, 87
File Control 103
file description
definition of 21
file descriptor . 21, 92, 94, 95, 99, 100, 104-106, 108-112, 119, 120, 123, 125, 127, 133-135, 137, 142-144, 147-149, 151, 154, 198, 203, 216, 217, 220, 347
duplicate 99
open stream 154
File Descriptor Deassignment 100
File Descriptor Manipulation 99
file group class
definition of 21
file mode
definition of 21
file offset
definition of 22
file other class
definition of 22
file owner class
definition of 22
file permission bits
definition of 22
file permissions 22
File Removal 86
file serial number
definition of 22
File Synchronization 107
file system 22, 23, 92, 95, 216, 271
read-only 23
filename
definition of 21
Files and Directories 302
definition of 82
first open (of a file)
definition of 22
for . . . . 1, i, viii, ix, 1-6, 9, 10, 19-85, 87-97, 100-152, 154, 157-223, 225-267, 269-315, 317-339, 341-343, 345, 347, 349
foreground 22
foreground process
definition of 22
foreground process group
definition of 22
foreground process group ID
definition of 22
fork()
function definition 48
Format of Directory Entries 82
FORTRAN ix
fpathconf()
function definition 96
fstat()
function definition 88
fsync()
function definition 107
ftruncate()
function definition 93
FULL . . . . . ix, 282-284
functions
_exit() 53
access() 89
aio_cancel() 141
aio_error() 138
aio_fsync() 147
aio_read() 115
aio_return() 140
aio_suspend() 144
aio_write() 121
chdir() 82
chmod() 89
chown() 92
close() 100
closedir() 82
creat() 84
dup() 99
dup2() 99
execl() 50
execle() 50
execlp() 50
execv() 50
execve() 50
execvp() 50
fchmod () 89
fcntl() 103
fdatasync() 110
fdopen() 154
fork() 48
fpathconf() 96
fstat() 88
fsync() 107
ftruncate() 93
getcwd() 83
kill() 65
link() 85
lio_listio() 128
lseek() 106
mkdir() 85
mkfifo() 85
mlock() 193
mlockall( ) 189
mmap() 197
mprotect() 207
mq_getattr() 291

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

mq_notify() 287
mq_open() 270
mq_receive() 285
mq_send() 282
mq_setattr() 289
mq_unlink() 280
msync() 211
munlock() 193
munmap() 205
nanosleep() 265
open() 83
opendir() 82
pause() 76
pipe() 99
read() 101
readdir() 82
rewinddir() 82
rmdir() 86
sched_get_priority_max() 240
sched_get_priority_min 240
sched_getscheduler() 238
sched_rr_get_interval() 240
sched_setparam() 229
sched_setscheduler() 234
sched_yield() 239
sem_close() 168
sem_destroy() 161
sem_getvalue() 184
sem_init() 158
sem_open() 162
sem_post() 180
sem_trywait() 172
sem_unlink() 170
sem_wait() 172
shm_open() 215
shm_unlink() 222
sigaction() 65
sigaddset() 65
sigdelset() 65
sigfillset() 65
sigismember() 65
sigpending 67
sigprocmask() 67
sigqueue() 72
sigtimedwait() 68
sigwaitinfo() 68
sleep() 76
stat() 88
sysconf() 78
timer_create() 253
timer_delete() 257
timer_getoverrun() 259
timer_gettime() 259
umask() 84
unlink() 86
utime() 93
waitpid() 53
write() 102
GA_macro_args
General Assertion Definition 4
GA_macro_result_decl
General Assertion Definition 4
ga_queuedAndRegularSignals
General Assertion Definition 58
GA_syncIODataIntegrityRead
General Assertion Definition 25
GA_syncIODataIntegrityWbeforeR
General Assertions 26
GA_syncIOFileIntegrityRead
General Assertion Definition 26
GA_syncIOFileIntegrityWrite
General Assertion Definition 26
ga_syncOpenWrite
General Assertion Definition 83
GA_upperLowerNames
General Assertion Definition 22
GA01 3
GA02 21
GA03 21
GD_CommonC_diffs
General Documentation Assertion definition 3
General 1, 293
definition of 1
General Assertions
ga_mqOpenMaxfd 270
ga_mqpctsOpenMaxfd 270
GA_portableFilenames 21
GA_sigqueueValue 57
GA_syncIODataIntegrityRead 25
GA_syncIODataIntegrityWbeforeR . 26
GA_syncIOFileIntegrityRead 26
GA_syncIOFileIntegrityWrite 26
ga_syncOpenWrite 83
m_ga_sigPendingQueued 58
M_GA_upperLowerNames 22
General Documentation Assertions
GD_CommonC_diffs 3
General File Creation 83
General Terms
definition of 20
Get Configurable Pathname Variables ... 96
Get Configurable System Variables 78
Get File Status 88
Get Message Queue Attributes 291
Get Scheduling Parameter Limits 240
Get Scheduling Parameters 232
Get Scheduling Policy 238
Get the Value of a Semaphore 184
Get Working Directory Pathname 83

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

getcwd()
function definition 83
GKS 343
Graphics Standards 343
group
change file 92
group ID
definition of 21, 22
effective 21
real 23
saved set- 24
supplementary 25
High Resolution Sleep 265
Historical Documentation and Introductory Texts 343
IAP 345
IEEE . 1, i, viii-xi, 1-5, 20, 21, 23, 28-38, 40-46, 48, 50, 53, 57, 63, 65, 67, 68, 76, 78, 81-89, 93, 96, 97, 99-107, 154, 160, 213, 225, 245, 246, 257, 269, 342, 343
IEEE 802.5 342
IEEE Draft Std P2003-199x 2
IEEE Std 100 343
IEEE Std 2003.1-1992 {4} 2
Implementation Conformance 3
Initialize an Unnamed Semaphore 158
Initialize/Open a Named Semaphore . . . 162
Input and Output 101
Input and Output Primitives 99, 306
int_max . . 159, 227-231, 234, 237, 245, 252, 265, 267
inter-process channel 99
IPC 99
IPC (see inter-process channel) 99
IRV 5
abbreviation 5
definition of 5
ISO 1539 342
ISO 1989 342
ISO 3166 343
ISO 639 343
ISO 7498 341
ISO 7942 343
ISO 8072 341
ISO 8326 341
ISO 8327 341
ISO 8348 341
ISO 8473 341
ISO 8571 341
ISO 8632 343
ISO 8649 342
ISO 8650 342
ISO 8652 342
ISO 8802-2 342
ISO 8802-3 342
ISO 8822 342
ISO 8823 342
ISO 8831 342
ISO 8832 342
ISO 8859-1 343
ISO 8907 343
ISO 9075 343
ISO 9127 343
ISO/IEC 10646 343
ISO/IEC 646 . . . . . 2, 5
ISO/IEC 8073 341
ISO/IEC 8802-4 342
ISO/IEC 9592 343
ISO/IEC 9899 . . . . . 2, 5
ISO/IEC 9945-2 343
IUT
Abbreviation 5
definition of 5
job control 22, 44, 56
definition of 22, 56
Job Control Signals 56
kill()
function definition 65
Language -Dependent Services for the C Programming Language 3
Language Standards 342
Language-Specific Services for the C Programming Language 154
definition of 154
Language-Specific Services for the C Programming Language 314
last close (of a file)
definition of 22
lifetime
session 25
link
definition of 22
link count
definition of 22
Link to a File 85
link()
function definition 85
lio_listio()
function definition 128
lio_nop
definition of 115
lio_nowait
definition of 115
lio_read
definition of 115
lio_wait
definition of 115

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

lio_write
definition of 115
List Directed i/o 128
Lock a Semaphore 172
Lock/Unlock a Range of Process Address Space 193
Lock/Unlock the Address Space of a Process 189
login
definition of 22
login name
definition of 22
lseek()
function definition 106
M_GA_macro_args
test specification macro definition ... 3
M_GA_macro_result_decl
test specification macro definition ... 4
m_ga_mqOpenMaxfd ()
test specification macro definition . 270
m_ga_mqpctsOpenMaxfd ()
test specification macro definition . 270
M_GA_portableFilenames
test specification macro definition .. 21
m_ga_queuedAndRegularSignals
test specification macro definition .. 58
m_ga_semOpenMaxfd
test specification macro definition . 157
m_ga_semPCTSOpenMaxfd
test specification macro definition . 157
M_GA_sigev_value()
test specification macro definition .. 57
m_ga_sigPending
test specification macro definition .. 58
m_ga_sigPendingQueued
test specification macro definition .. 58
M_GA_sigqueueValue()
test specification macro definition .. 57
M_GA_upperLowerNames
test specification macro definition .. 21
M_GD_CommonC_diffs
test specification macro definition ... 3
macro
feature test 21
Macros 5
Make a Directory 85
Make a fifo Special File 85
Manifest Constants 115, 246
Manipulate Signal Sets 65
map
definition of 22
Map Process Address to a Memory Object 197
map_failed
definition of 202
mask
file creation 84
MCL_CURRENT
definition of 190
MCL_FUTURE
definition of 190
Memory Locking Functions 189
Memory Management 187, 319
definition of 187
Memory Mapping Functions 197
memory object
definition of 22
Memory Object Synchronization 211
Memory Protection Signals 56
memory-resident
definition of 22
message
definition of 22
Message Passing 269
definition of 269
Message Passing Functions 270
message queue
definition of 22
MIT 344
mkdir()
function definition 85
mkfifo()
function definition 85
mlock
Reference Assertion definition 194
mlock()
function definition 193
mlockall( )
function definition 189
mmap
Reference Assertion definition 198
mmap()
function definition 197
mmap_ENOTSUP
Reference Assertion definition 199
mmap_SIGBUS
Reference Assertion definition 202
mode
definition of 22
mprotect
Reference Assertion definition 208
mprotect()
function definition 207
mprotect_ENOTSUP
Reference Assertion definition 208
mq_attr
C-language type definition 269
mq_close
Reference Assertion definition 279
mq_getattr

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Reference Assertion definition 291
mq_getattr()
function definition 291
mq_notify
Reference Assertion definition 288
mq_notify()
function definition 287
mq_notify_ebusy
Reference Assertion definition 289
mq_open
Reference Assertion definition 271
mq_open()
function definition 270
mq_open_eacces1
Reference Assertion definition 276
mq_open_eacces2
Reference Assertion definition 276
mq_open_eexit
Reference Assertion definition 276
mq_open_enoent
Reference Assertion definition 278
mq_open_max 9, 41, 42, 78-80, 277
mq_prio_max 41, 42, 78-80, 283, 284
mq_receive
Reference Assertion definition 285
mq_receive()
function definition 285
mq_receive_eagain
Reference Assertion definition 286
mq_send
Reference Assertion definition 282
mq_send()
function definition 282
mq_send_eagain
Reference Assertion definition 284
mq_send_einval
Reference Assertion definition 284
mq_send_emsgsize
Reference Assertion definition 284
mq_setattr
Reference Assertion definition 290
mq_setattr()
function definition 289
mq_unlink
Reference Assertion definition 280
mq_unlink()
function definition 280
mq_unlink_enoent
Reference Assertion definition 281
mqd_t
C-language type definition 269
ms_async
definition of 212
ms_invalidate
definition of 212
ms_sync
definition of 212
msync
Reference Assertion definition 211
msync()
function definition 211
msync_einval
Reference Assertion definition 213
MUMPS 342
munlock()
function definition 193
munlockall
Reference Assertion definition 191
munlockall ()
function definition 189
munlock
Reference Assertion definition 194
munmap
Reference Assertion definition 205
munmap()
function definition 205
munmap_SIGSEV
munmap_SIGSEV 205
name
login 22
user 27
name_max 9, 34, 41, 47, 167, 168, 171, 172, 221, 223, 278, 281
nanosleep()
function definition 265
Networking Standards 341
no_option . 3-5, 25-30, 34, 35, 37-40, 43, 45, 48-64, 66-75, 83, 84, 87, 89-97, 100-103, 105-152, 154, 157-223, 225-267, 269-292, 294-340
no_test . 24-30, 37, 38, 46, 48-54, 59, 60, 62, 63, 73, 75, 78, 79, 81, 84, 87, 88, 90-92, 94, 95, 100-103, 105, 106, 108-113, 115-128, 130-137, 139, 142-152, 157-161, 163, 165, 167-169, 171-197, 199-207, 210-215, 219, 221-223, 225-232, 234-237, 240, 242, 245, 246, 248-253, 255-258, 261-267, 269-281, 283, 284, 286-288, 290, 291, 294-339
no_test_support ... 24, 25, 48-54, 59, 60, 62, 63, 73, 75, 78, 79, 81, 84, 87, 88, 90-92, 94, 95, 105, 106, 109, 111-113, 115, 116, 118-128, 132-137, 139, 142-152, 157-161, 163, 167-169, 171-197, 199-207, 210-215, 221-223, 226-232, 234-237, 245, 246, 248-253, 255-258, 261-265,

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

267, 270-281, 283, 284, 286-288, 290, 291, 295-299, 301-305, 307-328, 330-334, 336-339
Normative References 2
Notify Process that a Message is Available on a Queue 287
null string
definition of 21, 22
o_creat 272
Open a File 83
Open a Message Queue 270
Open a Shared Memory Object 215
Open a Stream on a File Descriptor 154
open file
definition of 22
open file description
definition of 23
open()
function definition 83
open_max .. 9, 41, 42, 78-80, 105, 106, 157, 160, 167, 270, 277
opendir()
function definition 82
orphaned process group
definition of 23
Other C Language-Related Specifications . 3
Other C Language-Related Specifications . 3
Other Language-Related Specifications ... 4
Other Language-Related Specifications ... 4
Other Sources of Information 345
Other Standards 343
owner
change file 92
page
definition of 23
pagesize 7, 9, 42, 78-80, 194-197, 201, 205-207, 209, 211, 212, 215
parent directory
definition of 23
parent process
definition of 23
parent process ID
definition of 23
pass ... 3, 4, 21, 22, 24-40, 42-46, 48-64, 66-75, 78, 79, 81, 84, 87-97, 100-152, 154, 157-223, 225-267, 269-340
path prefix
definition of 23
path_max 9, 41, 167, 221, 277
pathname
absolute 20
definition of 23
get configurable values 96
get working directory 83
relative 24
pathname component
definition of 23
pause()
function definition 76
PCD.1b
abbreviation 5
definition of 5
pcts Symbols and Values 6
PCTS Variables 6
pcts-ftruncate
PCTS variable definition 12
PCTS_ . 4, 6-17, 24-27, 48-54, 58-64, 66, 67, 69-75, 84, 87, 88, 90-92, 94, 95, 100-102, 104-152, 157-188, 190-223, 226-240, 242-244, 248-267, 270-292
pcts_aio_cancel
pcts variable definition 11
pcts_aio_cancelable_ops 6, 142-144
PCTS variable definition 6
pcts_aio_error
PCTS variable definition 11
pcts_aio_fsync
PCTS variable definition 11
pcts_aio_max ... 6, 118, 119, 124, 125, 132, 151
PCTS variable definition 6
pcts_aio_max 132, 151
pcts_aio_read
PCTS variable definition 11
pcts_aio_return
PCTS variable definition 11
pcts_aio_suspend
PCTS variable definition 12
pcts_aio_write
PCTS variable definition 12
pcts_append_write_same_order 112
PCTS variable definition 6
pcts_chmod_sgid 90, 91
PCTS variable definition 6
pcts_chmod_suid 90
PCTS variable definition 6
pcts_clock-gettime
PCTS variable definition 12
pcts_clock_getres
PCTS variable definition 12
pcts_clock_settime
PCTS variable definition 12
pcts_defect_lockable_memory_limit_mlockall
PCTS variable definition 7
pcts_delaytimer_max 263
PCTS variable definition 6
pcts_detect_aio_error_aiocbp 141
PCTS variable definition 6
pcts_detect_enospc

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

PCTS variable definition 7
pcts_detect_eperm_clock_settime
PCTS variable definition 7
pcts_detect_invalid_flags_mmap
PCTS variable definition 7
pcts_detect_lockable_memory_limit_ mlock
PCTS variable definition 7
pcts_detect_message_data_corruption 7, 287
PCTS variable definition 7
pcts_detect_no_ap 193, 197
pcts_detect_not_multiple_of_pagesize 7, 197, 207, 211, 215
PCTS variable definition 7
pcts_detect_not_multiple_of_pagesize . 207
pcts_einval_fchmod
PCTS variable definition 7
pcts_extend_on_ftruncate
PCTS variable definition 8
pcts_fchmod
PCTS variable definition 12
pcts_fdatasync
PCTS variable definition 12
pcts_fsync
PCTS variable definition 12
pcts_gap_ 54
pcts_gap_clock_settime
PCTS variable definition 8
pcts_gap_mlock 206
PCTS variable definition 8
pcts_gap_mlockall 206
pcts variable definition 8
pcts_gap_modes_fchmod
PCTS variable definition 8
pcts_gap_mq-open
PCTS variable definition 8
pcts_gap_sched_setparam
PCTS variable definition 8
pcts_gap_sched_setscheduler
PCTS variable definition 9
pcts_gap_sem_init
PCTS variable definition 8
pcts_gap_sgid_fchmod
PCTS variable definition 8
pcts_gap_sigqueue
PCTS variable definition 8
pcts_gap_suid_fchmod
PCTS variable definition 8
pcts_get-priority_max
PCTS variable definition 12
pcts_get_priority_min
PCTS variable definition 12
pcts_gti_device . . 9, 26, 101, 102, 113, 115-117, 119-123, 125-127, 130, 132, 134-137, 148
PCTS variable definition 9
pcts_invalid_signal 75
PCTS variable definition 9
pcts_lio_listio
PCTS variable definition 13
pcts_map_fixed 9, 200, 201, 203-205
PCTS variable definition 9
pcts_map_private . . . 9, 200, 204, 206, 210, 212
PCTS variable definition 9
pcts_mlock
PCTS variable definition 13
pcts_mlockall
PCTS variable definition 13
pcts_mmap
PCTS variable definition 13
pcts_more_sa_siginfo_signals 63
PCTS variable definition 9
pcts_mprotect
PCTS variable definition 13
pcts_mq_as_file_type 87
PCTS variable definition 9
pcts_mq_close
PCTS variable definition 13
PCTS_MQ_FILE_DESCRIPTORS 270
pcts_mq_getattr
PCTS variable definition 13
pcts_mq_notify
PCTS variable definition 13
pcts_mq_open
PCTS variable definition 13
PCTS_MQ_OPEN_MAX 9
PCTS variable definition 9
pcts_mq_receive
PCTS variable definition 13
pcts_mq_send
PCTS variable definition 14
pcts_mq_setattr
PCTS variable definition 14
pcts_mq_unllink
PCTS variable definition 14
pcts_msync
PCTS variable definition 14
pcts_msync_storage
PCTS variable definition 14
pcts_multiple_of_pagesize . 9, 197, 201, 205-207, 211, 215
PCTS variable definition 9
pcts_munlock
PCTS variable definition 14
pcts_munlockall

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

PCTS variable definition 14
pcts_munmap
PCTS variable definition 14
pcts_name_max .. 9, 167, 171, 221, 223, 278, 281
PCTS variable definition 9
pcts_nanosleep
PCTS variable definition 14
pcts_no_sync_io_file ... 9, 84, 106, 109, 111
PCTS variable definition 9
pcts_open_max .. 9, 106, 157, 160, 167, 270, 277
PCTS variable definition 9
pcts_path_max 9, 167, 221, 277
PCTS variable definition 9
pcts_pipe_buf
PCTS variable definition 10
pcts_rap_clock_settime
pcts variable definition 10
pcts_rap_mlock
PCTS variable definition 10
pcts_rap_mlockall
PCTS variable definition 10
pcts_rap_mq_open
PCTS variable definition 10
pcts_rap_sched_setparam
PCTS variable definition 10
pcts_rap_sem_init
PCTS variable definition 10
pcts_rap_sgid_chmod
PCTS variable definition 10
pcts_rap_sgid_fchmod
pcts variable definition 10
pcts_rap_sigqueue
PCTS variable definition 10
pcts_read
PCTS variable definition 14
pcts_rofs 92
PCTS variable definition 10
pcts_sched_get_priority_max
PCTS variable definition 14
pcts_sched_get_priority_min
PCTS variable definition 15
pcts_sched_getparam
PCTS variable definition 15
pcts_sched_getscheduler
PCTS variable definition 15
pcts_sched_rr_get_interval
PCTS variable definition 15
pcts_sched_setparam
PCTS variable definition 15
pcts_sched_setscheduler
PCTS variable definition 15
pcts_sched_yield
PCTS variable definition 15
pcts_sem_close
PCTS variable definition 15
pcts_sem_destroy
PCTS variable definition 15
pcts_sem_ebusy 162
PCTS variable definition 10
pcts_sem_file_descriptors 157
pcts_sem_getvalue
PCTS variable definition 15
pcts_sem_init
PCTS variable definition 15
pcts_sem_invalid 186
PCTS variable definition 10
pcts_sem_is_fd 10, 87, 88, 160, 167
PCTS variable definition 10
pcts_sem_nsems_max 11, 53, 160, 169
PCTS variable definition 11
pcts_sem_nsems_max 169
pcts_sem_open
PCTS variable definition 16
pcts_sem_open_fd 163
pcts_sem_post
PCTS variable definition 16
pcts_sem_trywait
PCTS variable definition 16
pcts_sem_unlink
PCTS variable definition 16
pcts_sem_wait
PCTS variable definition 16
pcts_shm_as_file_type 11, 88
PCTS variable definition 11
pcts_shm_open
PCTS variable definition 16
pcts_shm_unlink
PCTS variable definition 16
pcts_sigqueue
PCTS variable definition 16
pcts_sigqueue_max 73
PCTS variable definition 11
pcts_sigtimedwait
PCTS variable definition 16
pcts_sigtimedwait_value 71
PCTS variable definition 11
pcts_sigwaitinfo
PCTS variable definition 16
pcts_timer_create
PCTS variable definition 16
pcts_timer_delete
PCTS variable definition 17
pcts_timer_getoverrun
PCTS variable definition 17
pcts_timer_gettime
PCTS variable definition 17
pcts_timer_max
PCTS variable definition 11

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

pcts_timer_settime
PCTS variable definition 17
pcts_unsupported_signal 75
PCTS variable definition 11
pcts_write
PCTS variable definition 17
Per-Process Timers 259
permission
file 22
persistence
definition of 23
PHIGS 343
Phrases 5
pipe . 7, 10, 23, 41, 55, 92, 99, 120, 125-127, 134, 136, 137, 306, 348
pipe()
function definition 99
Pipes 99
portable filename character set
definition of 23
posix.1 ..... 37, 343
POSIX.1b 2
Abbreviation 5
definition of 5
POSIX.1tm
Abbreviation 5
definition of 5
POSIX.tm
Abbreviation 5
definition of 5
preallocation
definition of 23
preempted process
definition of 23
priority
definition of 23
priority-based scheduling
definition of 23
privilege
definition of 23
process
background 20
child 20
controlling 20
create inter-process channel 99
definition of 23
delay execution 76
foreground 22
parent 23
send signal to 65
suspend execution 76
system 27
terminate 53
wait for termination 53
Process Creation 48, 53
Process Environment 78, 302
definition of 78
process group
background 20
definition of 23
foreground 22
orphaned 23
process group ID
definition of 23
foreground 22
process group leader
definition of 23
process group lifetime
definition of 23
process ID 23, 230, 233, 235, 238
definition of 23
parent 23
process lifetime
definition of 23
process list
definition of 23
Process Primitives 48, 295
definition of 48
prot_exec
definition of 199
prot_none
definition of 199
prot_read
definition of 199
prot_values
Reference Assertion definition 208
prot_write
definition of 208
Queue a Signal to a Process 72
R_10 ..... 137, 312
R_11 ..... 137, 312
R_12 137
R_2 . 25, 71, 92, 95, 111, 118, 124, 131, 140, 144, 146, 151, 159, 162, 169, 171, 176, 183, 186, 187, 192, 195, 202, 207, 214, 220, 223, 233, 236, 239, 240, 243, 251, 256, 258, 263, 266, 275, 279, 281, 283, 286, 289, 291-293, 301, 302, 305, 308-315, 317-320, 322-329, 331-334, 336-340
R_3 . 118, 127, 131, 141, 146, 177, 183, 187, 192, 196, 202, 209, 214, 220, 236, 243, 251, 263, 267, 275, 281, 283, 286, 289, 309-313, 318-325, 328, 329, 331, 333, 334, 336-339
R_4 . 120, 127, 131, 146, 177, 188, 196, 202, 209, 243, 251, 264, 276, 283, 309-

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

311, 313, 318, 319, 322, 323, 329, 331, 334, 336, 338
R_5 . 120, 127, 131, 188, 243, 251, 264, 276, 283, 309-311, 319, 330, 331, 334, 336, 338
R_6 . 121, 132, 189, 243, 251, 310, 311, 319, 330, 331
R_7 . . . . . 134, 311
R_8 . . . . . 134, 311
R_9 . . . . . 135, 311
Read from a File 101
read()
function definition 101
read-only file system
definition of 23
readdir()
function definition 82
real group ID
definition of 23
real user ID
definition of 23
Receive a Message From a Message Queue 285
Reference Assertions
clock_getres 249
clock_gettime 249
clock_settime 248
mlock 195
mlockall 190
mmap 199
mmap_ENOTSUP 199
mmap_SIGBUS 202
mprotect 208
mprotect_ENOTSUP 208
mq_close 279
mq_notify 288
mq_notify_ebusy 289
mq_open 271
mq_open_eacces1 276
mq_open_eacces2 276
mq_open_eexit 276
mq_open_enoent 278
mq_receive 285
mq_receive_eagain 286
mq_send 282
mq_send_eagain 284
mq_send_einval 284
mq_send_emsgsize 284
mq_setattr 290
mq_unlink 280
mq_unlink_enoent 281
msync 211
msync_einval 213
munlock 196
munlockall 191
munmap 205
munmap_SIGSEV 205
prot_values 199
sched_get_priority_max 242
sched_get_priority_min 242
sched_getparam 233
sched_getscheduler 238
sched_rr_get_interval 242
sched_setparam 229
sched_setscheduler 234
sched_yield 240
sem_close() 168
sem_destroy() 161
sem_getvalue() 184
sem_init() 158
sem_open() 162
sem_post() 180
sem_trywait() 172
sem_unlink() 170
sem_wait() 172
shm_exist_err 219
shm_open 219
shm_unlink 222
timer_create 253
timer_delete 258
timer_getoverrun 262
referenced shared memory object
definition of 24
region
definition of 24
regular file
definition of 24
Related Open Systems Standards 341
definition of 341
relative pathname
definition of 24
Remove a Directory 86
Remove a Message Queue 280
Remove a Named Semaphore 170
Remove a Shared Memory Object 222
Remove Directory Entries 86
Rename a File 86
Reposition Read/Write File Offset 106
Required Signals 55, 76
Requirements 3
Retrieve Error Status of Asynchronous i/o Operation 138
Retrieve Return Status of Asynchronous i/o Operation 140
rewinddir()
function definition 82
rmdir()
function definition 86
root directory
definition of 24

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

rtsig_max 78, 80
runnable process
definition of 24
running process
definition of 24
s_isgid 91
s_isuid 90
s_typeismq 87
s_typeissem 87
s_typeisshm 87
sa_siginfo 66
saved set-group-ID
definition of 24
saved set-user-ID
definition of 24
sc_rtsig_max
limit definition 80
SCHED_FIFO 226
definition of 226, 242
sched_get_priority_max
Reference Assertion definition 242
sched_get_priority_max(),
function definition 240
sched_get_priority_min
Reference Assertion definition 242
sched_get_priority_min()
function definition 240
sched_getparam
Reference Assertion definition 233
sched_getscheduler
Reference Assertion definition 238
sched_getscheduler()
function definition 238
SCHED_OTHER 228
definition of 226, 242
sched_param
C-language type definition 225
SCHED_RR
definition of 228, 242
sched_rr_get_interval()
function definition 240
sched_setparam
Reference Assertion definition 229
sched_setparam()
function definition 229
sched_setscheduler
Reference Assertion definition 234
sched_setscheduler()
function definition 234
sched_yield
Reference Assertion definition 240
sched_yield()
function definition 239
scheduling
definition of 24
Scheduling Parameters 225
Scheduling Policies 225
scheduling policy
definition of 24
Scope 1
seconds since the Epoch
definition of 24
sem_close
Reference Assertion definition 169
sem_close()
function definition 168
sem_destroy
Reference Assertion definition 162
sem_destroy()
function definition 161
sem_getvalue
Reference Assertion definition 185
sem_getvalue()
function definition 184
sem_init
Reference Assertion definition 159
sem_init()
function definition 158
sem_nsems_max 78, 79
sem_open
Reference Assertion definition 163
sem_open()
function definition 162
sem_post
Reference Assertion definition 181
sem_post()
function definition 180
sem_t
C-language type definition 157
sem_trywait()
function definition 172
sem_unlink()
function definition 170
sem_value_max 78, 79
sem_wait()
function definition 172
semaphore
definition of 24
Semaphore Characteristics 157
Semaphore Functions 158
semaphore lock operation
definition of 24
semaphore unlock operation:
definition of 25
sen_nsems_max 160, 169
Send a Message to a Message Queue ... 282
Send a Signal to a Process 65
serial number

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

file 22
session 341
definition of 25
session leader
definition of 25
session lifetime
definition of 25
Set File Access and Modification Times . 93
Set File Creation Mask 84
Set Message Queue Attributes 289
Set Scheduling Parameters 229
Set Scheduling Policy and Scheduling Parameters 234
SETUP 112
Shared Memory Functions 215
shared memory object
definition of 25
shm_exist_err
Reference Assertion definition 219
shm_open()
function definition 215
Reference Assertion definition 216
shm_unlink
Reference Assertion definition 222
shm_unlink()
function definition 222
si_user
definition of 61
sigabrt
definition of 55
sigaction()
function definition 65
sigaddset()
function definition 65
sigalrm
definition of 55
sigbus
definition of 56
sigchld
definition of 56
sigcont
definition of 56
sigdelset()
definition of 65
sigemptyset()
function definition 65
sigevent
C-language type definition 57
sigfillset()
function definition 65
sigfpe
definition of 55
sighup
definition of 55
siginfo_t
C-language type definition 61
sigint
definition of 55
sigismember()
function definition 65
sigkill
definition of 55
signal
concepts 55
definition of 25
examine and change action 65
examine and change blocked 67
examine pending 67
manipulate sets 65
mask 67
send to process 65
wait 68
Signal Actions 59
Signal Concepts 55
Signal Effects on Other Functions 65
Signal Generation and Delivery 57
Signal Names 55
Signals 55
sigpending()
function definition 67
sigpipe
definition of 55
sigprocmask()
function definition 67
sigqueue()
function definition 72
sigqueue_max 73
sigqueue_priv()
test specification macro definition .. 72
sigquit
definition of 55
sigrtmax
definition of 56
sigrtmin
definition of 56
sigsegv
definition of 55
sigstop
definition of 56
sigsuspend()
function definition 68
sigterm
definition of 55
sigtimedwait()
function definition 68
sigtstp
definition of 56
sigttin
definition of 56
sigttou

Copyright © 2003 IEEE – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

definition of 56
sigusr1
definition of 55
sigusr2
definition of 55
sigwaitinfo()
function definition 68
slash
definition of 25
sleep()
function definition 76
Special File Creation 85
Special Symbol clk_tck 81
standards
database 343
graphics 343
language 342
networking 341
open systems 341
stat()
function definition 88
stream
open 154
successfully transferred
definition of 25
supplementary group
definition of 25
supplementary group ID
definition of 25
supplementary groups 91
Suspend Process Execution 76
Synchronization 157, 314
definition of 157
Synchronize the Data of a File 110
Synchronize the State of a File 107
synchronized I/O completion
definition of 25
synchronized I/O data integrity completion
definition of 25
synchronized I/O file integrity completion
definition of 26
synchronized I/O operation
definition of 27
synchronous I/O operation
definition of 27
Synchronously Accept a Signal 68
system
definition of 27
system crash
definition of 27
System Databases 155, 314
definition of 155
system process
definition of 27
system reboot
definition of 27
System V 343
terminal
controlling 20
definition of 27
Terminate a Process 53
Terminology 20
Terminology and General Requirements 293
definition of 19
Terms 20
Test Methods 4
test specification macros
M_GA_macro_args 3
M_GA_macro_result_decl() 4
m_ga_mqOpenMaxfd () 270
m_ga_mqpctsOpenMaxfd () 270
M_GA_portableFilenames 21
m_ga_queuedAndRegularSignals .. 58
M_GA_semOpenMaxFD() 157
M_GA_semPCTSOpenMaxFD() .. 157
M_GA_sigev_value () 57
m_ga_sigPending 58
m_ga_sigPendingQueued 58
M_GA_sigqueueValue() 57
M_GA_stdC_proto_decl 5
M_GA_upperLowerNames 21
M_GD_CommonC_diffs 3
sigqueue_priv() 72
Time Value Specification Structures ... 245
timer
definition of 27
Timer Event Notification Control Block 246
Timer Operations 75
timer overrun
definition of 27
timer_abstime
definition of 247
timer_create
Reference Assertion definition 253
timer_create()
function definition 253
timer_delete
Reference Assertion definition 258
timer_delete()
function definition 257
timer_getoverrun
Reference Assertion definition 262
timer_getoverrun()
function definition 259
timer_gettime()
function definition 259

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

timer_max 78, 79
timer_settime()
    function definition 259
TR 10000 343
TRUE 6
Truncate a File to a Specified Length ... 93
TSG-1 Final Report 345
TSG-1 345
Type Definitions 246
Types of Conformance 3
uint_max 284
umask()
    function definition 84
undefined 52, 56
UNIX 343, 344
unlink()
    function definition 86
Unlock a Semaphore 180
Unmap Previously Mapped Addresses .. 205
unspecified 184, 185
USENIX 344
user ID
    definition of 27
    effective 21
    real 23
    saved set- 24
user name
    definition of 27
utime()
    function definition 93
VAX-11 345
Version 7 344
Wait for a Signal 68
Wait for Asynchronous i/o Request 144
Wait for Process Termination 53
waitpid()
    function definition 53
working directory
    change 82
Write to a File 102
write()
    function definition 102
X.25 342
X/Open 345
Yield Processor 239

Copyright © 2003 IEEE – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

--`,,`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

ISO/IEC 14515-1:2000/Amd.1:2003(E)
IEEE Std 2003.1b-2000
(Amendment to IEEE Std 2003.1-1992)

ICS 35.060

Price based on 363 pages

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860137 028 ■

INTERNATIONAL STANDARD

ISO/IEC 14515-1

IEEE Std 2003.1

First edition
2000-12-15

Information technology — Portable Operating System Interface (POSIX) — Test methods for measuring conformance to POSIX —

Part 1: System interfaces

Technologies de l'information — Interface de système de fonctionnement portable (POSIX) — Méthodes d'essai pour mesurer la conformité au POSIX —

Partie 1: Interfaces de système

This material is reproduced from ISO documents under International Organization for Standardization (ISO) Copyright License Number IHS/CC/1996. Not for resale. No part of these ISO documents may be reproduced in any form, electronic retrieval System or otherwise, except written consent of ISO (Case postal 56,1211 Geneva 20, Switzerland, FAX +41 22 734 10 79), IHS or the ISO Licensor's members.

Reference number
ISO/IEC 14515-1:2000(E)
IEEE
Std. 2003.1. 1992 edition
Reaffirmed 2000

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.ch
Web www.iso.ch

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

International Standard ISO/IEC 14515-1:2000(E)
IEEE Std 2003.1-1992 (Reaff 2000)

Information technology—Portable Operating System Interface (POSIX®)—Test methods for measuring conformance to POSIX—Part 1: System interfaces

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

Reaffirmed March 2000

IEEE-SA Standards Board

Approved November 2000

International Organization for Standardization
and by the
International Electrotechnical Commission



Adopted as an International Standard by the International Organization for Standardization and by the International Electrotechnical Commission





Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Abstract: ISO/IEC 14515-1:2000(E) (IEEE Std 2003.1-1992) provides a definition of the requirements placed upon providers of POSIX test methods for POSIX.1 (ISO/IEC 9945-1:1990; IEEE Std 1003.1-1990). These requirements consist of a POSIX.1-ordered list of assertions defining those aspects of POSIX.1 that are to be tested and the associated test methods that are to be used in performing those tests. This standard is aimed primarily at POSIX.1 test suite providers and POSIX.1 implementors. This standard specifies those aspects of POSIX.1 that shall be verified by conformance test methods.

Keywords: assertion, assertion test, base assertion, conditional feature, extended assertion, POSIX. POSIX Conformance Document, POSIX Conformance Test Procedure, POSIX Conformance Test Suite, test method, test result code

POSIX is a registered trademark of the Institute of Electrical and Electronics Engineers, Inc.

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2000 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published November 2000. Printed in the United States of America.

Print: ISBN 0-7381-2691-8 SH94894
PDF: ISBN 0-7381-2692-6 SS94894

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

International Standard ISO/IEC 14515-1:2000(E)

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 14515 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 14515-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces.*

ISO/IEC 14515 consists of the following parts, under the general title *Information technology — Portable Operating System Interface (POSIX) — Test methods for measuring conformance to POSIX*:

— *Part 1: System interfaces*

— *Part 2: Shell and utilities*

Annexes A and B form a normative part of this part of ISO/IEC 14515.



International Organization for Standardization/International Electrotechnical Commission
Case postale 56 • CH-1211 Genève 20 • Switzerland

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

IEEE Std 2003.1-1992 (Reaff 2000)

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

IEEE is the sole entity that may authorize the use of certification marks, trademarks, or other designations to indicate compliance with the materials set forth herein.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (978) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

(This introduction is not a normative part of IEEE Std 2003.1-1992, IEEE Standard for Information Technology—Test Methods for Measuring Conformance to POSIX—Part 1: System Interfaces, but is included for information only.)

This standard provides a definition of the requirements placed upon providers of a POSIX test method for the POSIX.1 {3} that are to be tested and the associated test methods that are to be used in performing those tests. This document is aimed primarily at POSIX.1 {3} test suite providers and POSIX.1 {3} implementors.this document specifies those aspects of the standard that shall be verified by conformance test methods.

Organization of This Standard

This document is organized into three portions:

1) Statement of scope, normative references, conformance requirements, and test methods (Section 1.)
2) Conventions and definitions (Section 2.)
3) Assertions to test POSIX.1 {3} interface facilities (Section 2. through 10.)

This introduction, any footnotes, notes accompanying the text, and the *informative* annexes are not considered part of this standard. Annexes A and B are informative.

Related Standards Activities

Activities to extend this stardard to address additional requirements are in progress, and similar efforts can be anticipated in the future.

The following areas are under active consideration at this time, or are expected to become active in the near future:[1]

1) Language-independent service descriptions of POSIX.1 {3}
2) C, Ada, and Fortran language bindings to (1)
3) Verification testing methods
4) Realtime facilities
5) Secure/Trusted System considerations
6) Network interface facilities
7) System Administration
8) Graphical User Interfaces
9) Profiles describing application- or user-specific combinations of Open Systems standards for: supercomputing, multiprocessor, and batch extension; transaction processing; realtime systems; and multiuser systems based on historical models
10) An overall guide to POSIX-based or related Open Systems standards and profiles

Extensions are approved as "amendments" or "revisions" to this document, following IEEE and ISO/IEC procedures.

Approved amendments are published separately until the full document is reprinted and such amendments are incorporated in their proper positions.

If you have an interests in participating in the TCOS working groups addressing these issues, pleasse send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, and ask to have this forwarded to the chairperson of the appropriate TCOS working group. If you have interest in participating in this work at the international level, contact your ISO/IEC national body.

[1] A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903; Telephone: +1 202 371-0101; FAX: +1 202 728-9614.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

IEEE Std 2003.1-1992 was prepared by the P2003.1 working group, sponsored by the Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society. At the time this standard was approved, the membership of the P2003.1 working group was follows:

Technical Commitee on Operating Systems and Application Environments (TCOS)

Chair:	Jehan-François Pâris

TCOS Standards Subcommittee

Chair:	Jim Isaak
Vice-Chairs:	Ralph Barker
	Hal Jespersen
	Lorraine C. Kevra
	Pete Meier
	Andrew T. Twigger
Treasurer:	Quin Hahn
Secretary:	Shane P. McCarron

P2003.1 Working Group Officials

Chair:	Roger J. Martin
Vice-Chairs:	Nicholas Ray Wilkes (1989–1990)
	Carol Raye (1987–1988)
Editor:	Anthony V. Cincotta
Secretaries:	Lowell G. Johnson (1989–1990)
	Doris Lebovits (1987–1988)

P2003.1 Technical Reviewers

Sanjay Agraharam	James M. Moe	Andrew T. Twigger
Anthony V. Cincotta	Anita Mundkur	Bruce Weiner
Steve Henderson	Gerald Powell	
Mark Lamonds	Ravi Tavakley	

P2003.1 Working Group

Sanjay Agraharam	Lorraine C. Kevra	Fred Noz
James P. Bound	Martin J. Kirk	Bob Palm
Kenneth R. Gibb	Mark Lamonds	Gerald Powell
Judy Guist	Robert M. Lenk	Ravi Tavakley
Rich Hamm	Kevin Lewis	Donn S. Terry
Steve Henderson	Martin J. McGowan III	Andrew T. Twigger
Scott Jameson	James M. Moe	Bruce Weiner
Greg Jones	Anita Mundkur	Brian Weis

The following persons provided valuable input during the balloting period:

Chuck Karish	Paul Rabin

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The following persons were members of the P2003.1 balloting that approved this standard for submission to the IEEE Standards Board:

Mike Lambert	*X/Open Organizational Representative*
Shane P. McCarron	*Unix International Organizational Representative*
Fritz Schulz	*OSF Organizational Representative*

David Athersych
Geoff Baldwin
Jerome E. Banasik
Robert Barned
Kabekode V.S. Bhat
Robert Bismuth
James Bohem
Robert Borochoff
Keith Bostic
James P. Bound
Phyllis Eve Bregman
A. Winsor Brown
Fred Lee Brown, Jr.
Nicholas A. Camillone
Clyde Camp
John Caywood
Kilnam Chon
Chan F. Chong
Anthony V. Cincotta
Robert L. Claeson
Richard Cornelius
William M. Corwin
Mike Cossey
William Cox
Donald W. Cragun
Ava Maria De Alvare
Terence Dowling
Stephen A. Dum
John D. Earls
Ron Elliott
Philip W. Enslow
Kenneth T. Faubel
Terence Fong
Kenneth R. Gibb
Michel Gien
Gregory W. Goddard
Dave Grindeland
Judy Guist
Charles E. Hammons
Allen Hankinson
Craig Harmer
Dale Harris
John L. Hill
David F. Hinnant
Lee A. Hollaar
Terrence W. Holm
Richard Hughes-Rowlands
Jim Isaak
Dan Iuster
Scott Jameson
Hal Jespersen
Lowell G. Johnson
Greg Jones
Lorraine C. Kevra
Jeffrey S. Kimmel
Martin J. Kirk
Dale L. Kirkland
Kenneth C. Klingman
D. Richard Kuhn
Robin B. Lake
Mark Lamonds
Doris Lebovits
Robert M. Lenk
David Lennert
Kevin Lewis
Joseph F. P. Luhukay
Robert J. Makowski
Roger J. Martin
Joberto S. B. Martins
Martin J. McGowan III
Robert W. McWhirter
Paul Merry
Doug Michels
James M. Moe
Anita Mundkur
Martha Nalebuff
Barry Needham
Fred Noz
John C. Penney
P. J. Plauger
Gerald Powell
Scott E. Preece
James M. Purtilo
Carol Raye
Christopher J. Riddick
Robert Sarr
Norman Schneidewind
Leonard W. Seagren
Karen Sheaffer
Dan Shia
Mukesh Singhal
Richard Sniderman
Douglas H. Steves
Scott A. Sutter
Robert Switzer
Ravi Tavakley
Donn S. Terry
Gary F. Tom
Andrew T. Twigger
Mark-Rene Uchida
Michael W. Vannier
John W. Walz
Alan G. Weaver
Bruce Weiner
Brian Weis
Peter J. Weyman
Andrew E. Wheeler, Jr.
Nicholas Ray Wilkes
Oren Yuen
Alaa Zeineldine
Jason Zions

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When the IEEE Standards Board approved this standard on December 3, 1992, it had the following membership:

Marco W. Migliaro, *Chair*
Donald C. Loughry, *Vice Chair*
Andrew G. Salem, *Secretary*

Dennis Bodson	Donald N. Heirman	T. Don Michael*
Paul L. Borrill	Ben C. Johnson	John L. Rankine
Clyde Camp	Walter J. Karplus	Wallace S. Read
Donald C. Fleckenstein	Ivor N. Knight	Ronald H. Reimer
Jay Forster*	Joseph Koepfinger*	Gary S. Robinson
David F. Franklin	Irving Kolodny	Martin V. Schneider
Ramiro Garcia	D. N. "Jim" Logothetis	Terrance R. Whittemore
Thomas L. Hannan	Lawrence V. McCall	Donald W. Zipse

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal	Richard B. Engelman	Stanley Warshaw
James Beall	David E. Soffrin	

Mary Lynne Nielsen
IEEE Standards Project Editor

With the recommendation and endorsement of the standard developers, this standard is dedicated to the Nicholas Ray Wilkes, who served as vice-chair of IEEE Std 1003.3-1991.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CLAUSE PAGE

1. General 1

1.1 Scope 1
1.2 Normative References 2
1.3 Conformance 2
1.4 Test Methods 4

2. Definitions and General Requirements 16

2.1 Conventions 16
2.2 Definitions 18
2.3 General Concepts 20
2.4 Error Numbers 23
2.5 Primitive System Data Types 24
2.6 Environment Description 24
2.7 C Language Definitions 25
2.8 Numerical Limits 26
2.9 Symbolic Constants 29

3. Process Primitives 31

3.1 Process Creation and Execution 31
3.2 Process Termination 42
3.3 Signals 50
3.4 Timer Operations 70

4. Process Environment 73

4.1 Process Identification 73
4.2 User Identification 74
4.3 Process Groups 81
4.4 System Identification 84
4.5 Time 85
4.6 Environment Variables 87
4.7 Terminal Identification 88
4.8 Configurable System Variables 91

5. Files and Directories 93

5.1 Directories 93
5.2 Working Directory 100
5.3 General File Creation 103
5.4 Special File Creation 119
5.5 File Removal 125
5.6 File Characteristics 137
5.7 Configurable Pathname Variables 157

6. Input and Output Primitives 165

6.1 Pipes 165
6.2 File Descriptor Manipulation 166
6.3 File Descriptor Deassignment 168

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CLAUSE PAGE

6.4 Input and Output 170
6.5 Control Operations on Files 178

7. Device- and Class-Specific Functions 184

7.1 General Terminal Interface 184
7.2 General Terminal Interface Control Functions 203

8. Language-Specific Services for the C Programming Language 217

8.1 Referenced C Language Routines 217
8.2 C Language Input/Output Functions 289
8.3 Other C Language Functions 292

9. System Databases 293

9.1 System Databases 293
9.2 Database Access 294

10. Data Interchange Format 298

10.1 Archive/Interchange File Format 298

Annex A (Informative) Bibliography 307

Annex B (Informative) Rationale and Notes 308

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

INTERNATIONAL STANDARD ISO/IEC 14515-1:2000(E)

Information technology—Portable Operating System Interface (POSIX®)—Test methods for measuring conformance to POSIX—Part 1: System interfaces

1. General

1.1 Scope

This standard defines the general requirements and test methods for measuring conformance to ISO/IEC 9945-1 :1990 (IEEE Std 1003.1-1990) hereinafter referred to as "POSIX.1" {3}.[1] It also defines the test assertions for measuring conformance to POSIX.1 {3}.

This standard is intended for use by

— Developers of POSIX.1 {3} test methods;
— Implementors of POSIX.1 {3} implementations;
— Application writers for POSIX.1 {3} conforming implementations;
— POSIX.1 {3} testing laboratories; and
— Others interested in validating the conformance of a vendor-claimed POSIX.1 {3} implementation

The purpose of this standard is to specify the test assertions and related test methods for measuring conformance of an implementation to POSIX.1 {3}.

Testing conformance of an implementation to a standard includes testing the claimed capabilities and behavior of the implementation with respect to the conformance requirements of the standard. These test methods are intended to provide a reasonable, practical assurance that the implementation conforms to the standard. Use of these test methods will not guarantee conformance of an implementation to POSIX.1 {3}; that normally would require exhaustive testing, which is impractical for both technical and economic reasons.

The technical specifications for a POSIX System Application Program Interface are defined in POSIX.1 {3}. IEEE Std 2003.1-1992 defines a means of measuring conformance to the POSIX.1 {3} technical specifications. Any question of

[1]The numbers in curly braces correspond to those of the references in 1.2.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

interpretation of those technical specifications arising from the use of this standard is a question of interpretation of POSIX.1 {3}.

1.2 Normative References

The following standards contain provisions that, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

{1}ISO/IEC 646: 1991 [2] *Information technology—ISO 7-bit coded character set for information interchange.*

{2}ISO/IEC 9899: 1990 *Information technology—Programming languages—C.*

{3}ISO/IEC 9945-1:1990 (IEEE Std 1003.1-1990) [3] *Standard for Information Technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].*

{4}IEEE Std 1003.3-1991 *IEEE Standard for Information Technology—Test Methods for Measuring Conformance to POSIX.*

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

NOTE — The types of assertion numbering used in this document are explained in IEEE Std 1003.3-1991 {4}.

D01(A) The conformance document defines an environment in which an application can be run with the behavior specified by POSIX.1 {3} in 1.3.1.1 of the PCD.1[4].

1.3.1.2 Documentation

D02(A) The conformance document contains a statement that indicates the full name, number, and date of the POSIX.1 {3} standard that applies in 1.3.1.2 of the PCD.1.

D03(C) If the implementation makes international software standards available to a conforming application and the implementation documents this:

The list of these standards is contained in 1.3.1.2 of the PCD.1.

NOTE — The conformance document requirements are specified in 1.4.17.1.

1.3.2 Application Conformance

There are no test methods provided for this subclause.

1.3.3 Language-Dependent Services for the C Programming Language

D04(C) If the C-language provided does not conform to the C Standard {2}:

[2]ISO/IEC documents can be obtained from the ISO office, 1, rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse.
[3]IEEE documents can be obtained from the The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, New York 10017, USA.
[4]Abbreviations are defined in 1.4.2.3.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The conformance documentation states that the C-language provided does not conform to the C Standard {2} in 1.3.3 of the PCD.1.

1.3.3.1 Types of Conformance

There are no test methods provided for this subclause.

1.3.3.2 C Standard Language-Dependent System Support

D05(C) If the C Standard {2} language binding is supported:
The conformance document states the version of the C Standard {2} referenced in fulfilling the requirements of Section 8. of POSIX.1 {3} in 1.3.3.2 of the PCD.1.

1.3.3.3 Common-Usage C Language-Dependent System Support

DGA01 For all functions stated in Section 8.1 of POSIX.1 {3}:
The details of all differences between the interface provided and the interface that would have been provided had the C Standard {2} been implemented instead of Common-Usage C are contained in 8.1 of the PCD.1.

D06(C) If the Common-Usage C language binding is supported and differences exist from the C Standard {2}:
The version of the C Standard {2} used for referencing the differences is stated in 1.3.3.3 of the PCD.1.

1.3.4 Other C Language-Related Specifications

NOTE — When a macro exists for any function, except *assert*(), *setjmp*(), and *sigsetjmp*(), then all assertions, except those in the Synopsis clause, are tested for both the macro and the function interfaces. The identifiers with external linkage providing access to *setjmp*() and *sigsetjmp*() shall be tested. Assertions in the Synopsis clause are specifically related to the function or macro interfaces as appropriate. (See 1.4.8.)

GA01 For all elements except *abort*(), *assert*(), *getc*(), *putc*(), *setjmp*(), *sigsjmp*(), and *tzset*():
If the function is defined as a macro:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary.

For *funct*() of *setjmp*() and *sigsetjmp*():
If *funct*() is defined as a macro:
It protects its result value with extra parentheses when necessary.

There are no assertions specific to this subclause.

1.3.5 Other Language-Related Specifications

There are no test methods provided for this subclause.

1.3.6 Test Methods Conformance

1.3.6.1 Requirements

A conforming implementation of POSIX.3.1 test methods shall meet all of the following criteria:

1) All test method requirements of 1.4.4 through 1.4.19 that are associated with required base assertions shall be implemented.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2) If the test method tests for a specific POSIX.1 {3} profile, then all test method requirements of 1.4.4 through 1.4.19 that are associated with the requirements of this profile shall be implemented.
3) All required base assertions shall be tested.
4) If the PCTS.1 tests for a specific POSIX.1 {3} profile, then all conditional base assertions associated with the requirements of this profile shall be implemented.
5) A conforming PCTS.1 shall conform to the requirements of IEEE Std 1003.3-1991 (POSIX.3 {4}).
6) A conforming PCTS.1 shall document the limitations on POSIX.1 {3} testing when employing the PCTS.1. *Example:* This PCTS.1 considers appropriate privilege to be implemented monolithically as a single privilege.

1.3.6.2 Recommendations

A conforming implementation of POSIX.3.1 test methods should meet the following criterion:

— If extended assertions are within the POSIX.1 {3} profile being tested, then these assertions should, when possible, be tested.

1.4 Test Methods

1.4.1 Introduction

This clause defines the test methods to be used to measure conformance to POSIX.1 {3} and contains requirements and recommendations for the test methods specific to testing POSIX.1 {3}.

The contents of POSIX.3.1 are C-language specific because POSIX.1 {3} is specified in terms of C-language bindings. Except where explicitly stated, the requirements and recommendations of POSIX. 3.1 apply to a PCTS.1 written in either Common-Usage C or C Standard {2} and test conformance to either Common-Usage C or C Standard {2} language bindings of a target system.

The test methods used for POSIX.1 {3} conformance measurement shall consist of a PCTS.1 and a documentation audit of the POSIX Conformance Document as required by POSIX.1 {3} and its associated PCTP. 1.

1.4.2 Definitions

1.4.2.1 Terminology

1.4.2.1.1 may: An option for *test methods.*

In this standard the negative of *may* is *need not.*

1.4.2.1.2 shall: A requirement for *test methods.*

1.4.2.1.3 should: A recommended practice for *test methods.*

1.4.2.2 General Terms

1.4.2.2.1 assertion: A statement of functionality or behavior for a POSIX *element* that is derived from the POSIX standard being tested and that is true for a conforming POSIX implementation. [IEEE Std 1003.3-1991]

1.4.2.2.2 assertion number: The numeric identifier assigned to an *assertion.*

The name of the *element* and the *assertion number* together uniquely identify an *assertion.* [IEEE Std 1003.3-1991]

1.4.2.2.3 assertion test: The software or procedural methods that ascertain the conformance of a POSIX implementation to an *assertion.* [IEEE Std 1003.3-1991]

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

1.4.2.2.4 base assertion: An *assertion* that is required to be tested for required features and for implemented *conditional features*. [IEEE Std 1003.3-1991]

1.4.2.2.5 conditional feature: A feature or behavior referred to in a POSIX standard that need not be present on all conforming implementations. [IEEE Std 1003.3-1991]

1.4.2.2.6 development system: The computer system used to compile and configure a PCTS.1. [IEEE Std 1003.3-1991]

1.4.2.2.7 element: A functional interface or a namespace allocation.

Examples of elements are C functions or utility programs. Examples of namespace allocation include headers or error return value constants. [IEEE Std 1003.3-1991]

1.4.2.2.8 extended assertion: An *assertion* that is not required to be tested. [IEEE Std 1003.3-1991]

1.4.2.2.9 modem control: The monitoring of modem status lines.

1.4.2.2.10 POSIX Conformance Document (PCD): The conformance document required by a POSIX standard. [IEEE Std 1003.3-1991]

1.4.2.2.11 POSIX Conformance Test Procedure (PCTP): The nonsoftware procedures possibly used in conjunction with other test methods to measure conformance. [IEEE Std 1003.3-1991]

1.4.2.2.12 POSIX Conformance Test Suite (PCTS): The collection of software possibly used in conjunction with other test methods to measure conformance. [IEEE Std 1003.3-1991]

1.4.2.2.13 required feature: Either a single facility or behavior, or one of a pair of alternative facilities or behaviors, required by a POSIX standard that is always present on a conforming implementation. [IEEE Std 1003.3-1991]

1.4.2.2.14 target system: The combination of the computer system on which the PCTS is executed and the parts of the *development system* that are used to generate the executable code of a PCTS. [IEEE Std 1003.3-1991]

1.4.2.2.15 test method: The software, procedures, or other means specified by a POSIX standard to measure conformance.

Test methods may include a PCTS, PCTP, or an audit of a PCD. [IEEE Std 1003.3-1991]

1.4.2.2.16 test result code: A value that describes the result of an *assertion test*. [IEEE Std 1003.3-1991]

1.4.2.2.17 unattainable limit: A limit that is undefined for a *target system* or that has a magnitude exceeding the POSIX.1 {3} specified minimum and that would require an unreasonable amount of time or system resources to test.

1.4.2.3 Abbreviations

For the purposes of this standard, the following abbreviations apply:

1.4.2.3.1 C Standard: ISO/IEC 9899 , *Information technology—Programming languages—C {2}.*

1.4.2.3.2 Common-Usage C: Common-Usage C Language-Dependent System Support.

1.4.2.3.3 PCD.1: POSIX Conformance Document for POSIX.1 {3}.

1.4.2.3.4 PCTP. 1: POSIX Conformance Test Procedures for POSIX.1 {3}.

1.4.2.3.5 PCTS.1: POSIX Conformance Test Suite for POSIX.1 {3}.

1.4.2.3.6 POSIX: Colloquial name for the collection of IEEE 1003 standards, draft standards, and projects.

1.4.2.3.7 POSIX.1: ISO/IEC 9945-1 :1990 (IEEE Std 1003.1-1990).

1.4.2.3.8 POSIX.3: IEEE Std 1003.3-1991.

1.4.2.3.9 POSIX.3.1: The POSIX.3 standard that defines the test methods for the POSIX.1 {3} standard; a reference to this standard.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

1.4.3 Assertion Concepts

1.4.3.1 Security Features

The assertions in this standard, in general, do not address the conditional features associated with extended security controls, alternate file access control mechanisms, and additional file access control mechanisms.

1.4.3.2 Appropriate Privileges

All assertions that require either the presence or absence of appropriate privilege state their requirements. Assertions that contain no stated requirement relating to appropriate privilege hold true regardless of what appropriate privileges the calling process has.

1.4.3.3 Testing Constraints

A testing constraint is a conditional feature, behavior, or additional hardware or software needed to test an assertion. Testing constraints are employed within the classification of the assertion with the syntax:

(testing constraint?classification:UNTESTED)

1.4.3.4 PCD Announcement Mechanism

When assertions are dependent on the PCD.1 for an announcement mechanism about the support or nonsupport of POSIX.1 {3} conditional features, then the conditional phrase of the assertion shall incorporate the appropriate PCD symbol (see Table 1.4).

1.4.4 PCTS Execution

1.4.4.1 PCTS Target System Support Facilities

Support facilities are required to establish an operational environment for some tests within a PCTS.1, where it is necessary to undertake tasks that are beyond the scope of POSIX.1 {3}. The implementation of these support facilities will differ between different target systems, and it will be necessary for the user of the PCTS.1 to provide an interface to the support facilities to be able to execute the PCTS.1 fully.

1.4.4.1.1 Requirements

The interface syntax for a target system support facility shall be specified in the PCTS.1 documentation. The values designating successful and unsuccessful provision of the requested facility shall also be specified in the PCTS.1 documentation. A target system support facility shall return an unsuccessful indicator if the target system does not support the facility.

1.4.4.1.2 Recommendations

Where POSIX.1 {3} does not provide the means for establishing the required operational environment, a portable PCTS.1 should not make assumptions about the functions a target system will use to establish this environment. To establish the test environment, the PCTS.1 should use well-defined target system support facilities that may have to be provided by the user of the PCTS.1.

To increase portability, a PCTS.1 should allow the test installer to provide target system support facilities to establish an environment for testing.

A target system support facility should be used to control

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— Associating a controlling terminal with a process
— Establishing appropriate privileges for a process
— Mounting a file system with read and write capabilities
— Mounting a file system with read only capability
— Unmounting a file system
— Setting S_ISUID mode with *chmod*()
— Setting S_ISGID mode with *chmod*()

The PCTS.1 should not require information about the manner in which the implementation provides the requested facilities nor about the time taken to provide these facilities.

1.4.4.2 Test Tolerances

1.4.4.2.1 Requirements

In order to test for proper functioning of time-related functions, a PCTS.1 shall permit allowances for a degree of variability in the time to perform certain operations. For example, comparison of the times returned by the *times*() function to the times returned separately by the child processes will vary slightly between separate executions of the test.

POSIX.1 {3} does not define the timeliness or the asynchronous nature of certain events. The PCTS.1 shall expect that the effect of an event on both the initiating and all target processes shall be completed in a reasonable amount of time and that all of the affected processes shall continue in the expected manner after the event has occurred. The PCTS.1 shall report or document the time variation that it allows for a test tolerance.

1.4.4.2.2 Recommendations

None.

1.4.4.3 Timeout Tolerances

1.4.4.3.1 Requirements

In order to test certain time-related events, a PCTS.1 shall permit allowances for variance in the time at which the timer goes off. For instance, the *alarm*() function sets a timer that will cause a SIGALRM to be sent at the specified time. Variations in system load and process scheduling can cause the actual time at which SIGALRM is sent to vary.

1.4.4.3.2 Recommendations

A PCTS.1 should use a user configuration variable to specify the defined amount of time beyond the specified timer time that a test should wait before reporting failure.

1.4.5 PCTS Testing Constraints

1.4.5.1 Requirements

The symbols to be used by a PCTS.1 when denoting POSIX.3.1 testing constraints are specified in Table 1.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Table 1.1—Testing Constraints

Symbol	Testing Constraint
PCTS_APP_LINK_DIR	The implementation permits the appropriate privilege process to create links to directories.
PCTS_GTI_DEVICE	The implementation provides device types that support the general terminal interface.
PCTS_GTI_BUFFERS_OUTPUT	The general terminal interface device buffers its output.
PCTS_PROCESS_LIMIT	The system process limit is not exceeded before {CHILD_MAX} is reached.
PCTS_CHMOD_SET_IDS	The function *chmod*() allows setting the S_ISUID and S_ISGID modes.
PCTS_INVALID_SIGNAL	An invalid or unsupported signal number exists.
PCTS_INVALID_OWNER	An invalid group or owner ID exists.
PCTS_APP_MODE	The implementation permits the appropriate privilege process to change file mode.
PCTS_APP_OWNER	The implementation permits the appropriate privilege process to change owner and group IDs.
PCTS_APP_TIMES	The implementation permits the appropriate privilege process to change file times.
PCTS_ROOT_WRITABLE	The root file system is not read-only.
PCTS_HPA_FILE	The implementation supports the high-performance attribute file.

1.4.5.2 Recommendations

None.

1.4.6 PCTS Limits and Configuration Variables

1.4.6.1 Requirements

Table 1.2 defines sets of limits. The POSIX.1{3} values {ARG_MAX}, {CHILD_-MAX}, {LINK_MAX}, {MAX_CANON}, {MAX_INPUT}, {NAME_MAX}, {OPEN_MAX}, {PATH_MAX}, {PIPE_BUF}, and {TZNAME_MAX} used by the PCTS.1 shall be obtained from the *pathconf*() and *sysconf*() calls, unless the assertion explicitly states that the value is to be obtained from a specific header. The POSIX.1 {3} limit value for {PCTS_LOCK_ MAX} is imposed by the system.

Because certain of these POSIX.1 {3} limits, for practical testing purposes, may be unattainable or require an unreasonable amount of time or resources before they are attained, values for the mimimurn PCTS.1 testing limit (PCTS_mtl) are provided as specified in Table 1.2. When a specific PCTS_mtl is less than the associated POSIX.1 {3} limit, a PCTS.1 may set the actual test limit value (PCTS_tlv) such that PCTS_mtl is less than or equal to PCTS_tlv and PCTS_tlv is less than or equal to the POSIX.1 {3} limit. If the implementation does not provide a POSIX.1 {3} limit, the PCTS.1 shall set the test limit value such that PCTS_mtl is less than or equal to PCTS_tlv.

When the POSIX.1 {3} limit obtained from *pathconf*(), *syscon*(), or another system-imposed limit is less than or equal to the associated PCTS_mtl, the POSIX.1 {3} limit shall be used for testing, as shown in Table 1.2.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Table 1.2—Configuration Variables

PCTS Limit Name	POSIX.1 Limit	PCTS_mtl (PCTS.1 minimum test limit)
PCTS_ARG_MAX	{ARG_MAX}	The lesser of {ARG_ MAX}, as obtained from *sysconf*(), and ten times {_POSIX_ARG_MAX}.
PCTS_CHILD_MAX	{CHILD_MAX}	The lesser of {CHILD_MAX}, as obtained from *sysconf*(), and 256.
PCTS_LINK_MAX	{LINK_MAX}	The lesser of {LINK_MAX}, as obtained from *pathconf*(), and 256.
PCTS_LOCK_MAX	system-imposed	The lesser of the system-imposed limit and 2500.
PCTS_MAX_CANON	{MAX_CANON}	The lesser of {MAX_CANON}, as obtained from *pathconf*(), and four times {_POSIX_MAX_CANON}.
PCTS_MAX_INPUT	{MAX_INPUT}	The lesser of {MAX_INPUT}, as obtained from *pathconf*(), and four times {_POSIX_MAX_INPUT}.
PCTS_NAME_MAX	{NAME_ MAX}	The lesser of {NAME_MAX}, as obtained from *pathconf*(), and 2048.
PCTS_OPEN_MAX	{OPEN_MAX}	The lesser of {OPEN_MAX}, as obtained from *sysconf*(), and 256.
PCTS_PATH_MAX	{PATH_MAX}	The lesser of {PATH_MAX}, as obtained from *pathconf*(), and 4096.
PCTS_PIPE_BUF	{PIPE_BUF}	The lesser of {PIPE_BUF}, as obtained from *pathconf*(), and 32767.
PCTS_TZNAME_MAX	{TZNAME_MAX}	The lesser of {TZNAME_MAX}, as obtained from *sysconf*(), and 256.

The term *obtained from* in Table 1.2 refers to the meaning of the return value from a call to *pathconf*() or *sysconf*(). If this return value is −1 and the error indicator is unchanged, the PCTS.1 shall interpret this as a value greater than the corresponding PCTS_tlv.

The manner in which the assertion is modified is described in each of the assertions affected.

A PCTS.1 shall use configuration parameters to provide information about the control sequences used to perform specific terminal operations, as shown in Table 1.3. The sequences that an implementation uses to erase characters and lines differ from implementation to implementation. A PCTS.1 can assume no knowledge of these character sequences. The PCTS.1 will verify that the character sequence used to erase the character/line matches the expected sequence for the character/line erase operation for the target system. These character sequences should be provided to the PCTS.1 using the configuration variables specified in Table 1.3.

Table 1.3—Configuration Control Sequences

Test Option	Value
PCTS_ECHOE	Erase character sequence, or empty if none.
PCTS_ECHOK	Erase line sequence, or empty if none.

The testing of character and line erase sequences is undertaken in canonical mode. The PCTS.1 shall set up the test for character erasure by first ensuring that the terminal input line is empty and then placing a sequence of characters into the input line. The PCTS.1 shall then issue an erase character command and ensure that the sequence specified in {PCTS_ECHOE} is used to erase the character. A similar technique is used to test for line erasure except the sequence specified in {PCTS_ ECHOK} is used to check that the correct sequence of characters is used to erase the line.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

1.4.6.1.1 Recommendations

None.

1.4.7 Test Methods for Headers

1.4.7.1 Requirements

The symbol _POSIX_SOURCE shall be defined by the PCTS.1 prior to the inclusion of any headers.

Since headers are not required to be readable files of C source code on the target system, it is necessary to use indirect ways to test their content. The items defined in headers shall be tested as follows to establish compliance as required in POSIX.1 {3}.

— *Symbolic constants*, *macros*, and *types* shall be tested for existence and their functional correctness. See 1.4.8 for more requirements on macros.
— *Function prototypes* in the C Standard {2} shall be tested for their existence and correctness of definition.
— *Function declarations* in Common-Usage C shall be tested for their existence and correctness of definition.
— *Function return values* in Common-Usage C for functions not returning integer type shall be tested for correctness of return value.
— *Data aggregate definitions* shall be tested for their existence and correctness of all members and types associated with members that are specified in POSIX.1 {3}.
— *Variables* shall be tested for their existence and types.
— The ability to include a header more than once shall be tested.
— The ability to access a symbol whose definition is required in more than one header shall be tested.

1.4.7.2 Recommendations

Headers should be tested by including them in test programs. The following criteria should be used for type checking.

— For a definition of a structure type, the size of each structure member should be compared to the size of an item of the type of the member or to the size of the type of the member itself.
— For a variable declaration, the size of the variable should be compared to the size of a variable of the specified type or to the size of the specified type itself. This test applies to members of structures, not to structures themselves.
— For a variable, the variable should be compared to another variable of the same type. This test applies to members of a structure, not to structure variables.
— For a variable, the variable should be assigned to another variable of the same type.
— For a variable, the address of the variable should be assigned to another variable of the type pointer to the defined type. This does not apply to function return values.

The detection of a nonconformity relies upon the C compiler generating an error or warning message. If such a message is generated, then there should be either a PCTP.1 to check the compiler output to resolve whether the tests **PASS** or **FAIL**, or software to do the checking automatically.

To test for proper behavior of duplicate header inclusion and duplicate symbolic definition in headers, a test should include all of the headers to be tested in a valid order such that there are duplicate inclusions of all headers.

If the PCTS.1 uses a language compiler or translator to accomplish testing of the assertions related to headers, the PCTS.1 should not rely solely on the return value from that compiler or translator, but should also examine any messages produced during test execution.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

1.4.8 Test Methods for Macros

1.4.8.1 Requirements

A PCTS.1 shall test that any interface that is implemented as a macro has an underlying function of the same name (if required by the standard) and that both the macro and the function behave in the same manner with respect to the assertions.

The PCTS.1 shall also test that the macro is implemented in such a manner that it evaluates its arguments only once, fully protected by parentheses when necessary, and yields a resulting value that can be used in complex expressions without requiring extra parentheses.

1.4.8.2 Recommendations

None.

1.4.9 Test Methods for Invalid Constant Values

1.4.9.1 Requirements

Some POSIX.1 {3} functions take a symbolic constant as an argument and have a specified behavior when the argument has an invalid value. Because an implementation can have extensions that permit the function to accept values other than those specified by POSIX.1 {3}, a portable PCTS.1 shall not simply choose such a value.

1.4.9.2 Recommendations

A PCTS.1 should use a user configuration variable to select an invalid value of each constant for which the behavior of invalid values is tested.

1.4.10 Test Methods for *errno* Checking

1.4.10.1 Requirements

A test shall be performed for detection of all values of *errno* specified in the Errors subclause of each element. If more than one error occurs in processing a function call, an assertion test shall not depend on the order in which the errors are detected or in which error code is returned.

Tests for error code detection shall be coded to generate only a single error code when possible.

1.4.10.2 Recommendations

None.

1.4.11 Test Methods for File Accessibility

1.4.11.1 Requirements

POSIX.1 {3} defines the cases in which access is granted to a file from a POSIX.1 {3} interface both in the case that the implementation provides a mechanism for a process to exist with the appropriate privilege to override the standard file access control mechanism and the case that the process has standard file access capabilities. A PCTS.1 shall test for the accessibility of a file in both of these cases and shall test for the [EACCES] error indication in the case that standard file access is denied to a process. These tests shall be carried out for each of the interfaces that contain assertions refering to the ability to access a file or assertions refering to the [EACCES] error.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

POSIX.1 {3} defines the cases of additional and alternate file access control mechanisms without defining the manner in which these mechanisms can be activated. A PCTS.1 is not required to test these mechanisms, but shall run in an environment where these mechanisms exist but are not enabled.

1.4.11.1.1 Recommendations

None.

1.4.12 Test Methods for Signal Functions

1.4.12.1 Reentrance of Interrupted Functions

1.4.12.1.1 Requirements

A PCTS.1 shall test that, when a function in a quiescent state is interrupted by a signal, a call to the interrupted function from the signal handling routine behaves correctly and does not improperly affect the result of the interrupted function.

1.4.12.1.2 Recommendations

None.

1.4.12.2 Functions Interruptable by Signals

1.4.12.2.1 Requirements

When executing a test to verify that *errno* is set to [EINTR] when a function is interrupted, a PCTS.1 shall not use the default action of job control signals to effect the interruption. These signals cause the process to stop and do not actually interrupt the function.

1.4.12.2.2 Recommendations

None.

1.4.12.3 *sigaction*() Testing

1.4.12.3.1 Requirements

See *sigaction*() assertions.

1.4.12.3.2 Recommendations

If the behavior associated with {_POSIX_JOB_CONTROL} is supported, the recommended test method is to establish a known value for a globally accessible variable and then to call *sigaction*() for the SIGSTOP signal to attempt to associate a signal catching function with this signal. The handler would be written to change the known value. A SIGSTOP signal should then be generated, followed by a SIGCONT signal to cause the process to continue. If the known value has changed, the handler was established even though *sigaction*() failed.

If the behavior associated with {_POSIX_JOB_CONTROL} is not supported, the recommended method for testing that the signal catching function is not associated with the signal when *sigaction*() fails is to install a signal-catching function for SIGKILL that changes the value of a global variable. Then a SIGKILL signal is sent to the process. If the process can check whether the global variable has been changed (even determining that it has not), SIGKILL has not been delivered properly.

Possible tests include

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— Having the handler do an *_exit(n)* and having the parent check exit status
— Writing to the file system from the handler

1.4.13 Test Methods for Files and Input/Output Primitives

1.4.13.1 Requirements

A PCTS.1 shall use at least one of the file types that are defined in POSIX.1 {3} when testing an assertion that applies to files. The file type chosen may differ for different assertions. Wherever a specific set of file types are defined in an assertion, a PCTS.1 shall use each of the file types stated.

1.4.13.2 Recommendations

A PCTS.1 implementor may choose to improve test coverage by increasing the number of file types covered by assertions.

1.4.14 File Descriptor Closure Test Method

1.4.14.1 Requirements

A PCTS.1 shall test that file descriptors are closed by *_exit*(), *exit*(), and *abort*(). (See GA41 in 3.2.)

1.4.14.2 Recommendations

The recommended method for testing file descriptor closure on a regular file is as follows:

— Parent process opens a file, then creates a child with *fork*().
— The child process creates an exclusive lock on the file and calls *_exit*().
— Upon termination of its child process, the parent attempts to exclusively lock the same part of the file. If it can, the test passes. If it cannot, the test fails.

The recommended method for testing this on a pipe (or FIFO) is as follows:

1) Create a pipe.
2) Fork a child process.
3) Close the write channel of the pipe in the parent process.
4) Terminate the child process.
5) Expect a *read*() on the pipe in the parent process to return a value of 0.

1.4.15 Test Methods for Terminal I/O Functions

1.4.15.1 Requirements

Terminal I/O testing shall not be performed using an interactive test method, and the implementation shall provide a method whereby a process can acquire a controlling terminal.

1.4.15.2 Recommendations

Terminal I/O should be tested using an automated test method. This method provides for accuracy and repeatability of test results by eliminating the need for human intervention while running a PCTS.1. An automated method of terminal I/O testing requires two asynchronous ports to be electrically connected in closed-loop mode and for these ports to support modem control protocols. One of the closed-loop ports is the driver port and the other is the target port. The terminal I/O tests send character data between the driver port and the target port via the connecting cable as though the driver port were a terminal. The tests then compare the data received with the expected results.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Target systems normally configured with less than two asynchronous ports may be specifically configured with two asynchronous ports for the terminal I/O tests. The normal configuration will be considered to have passed the terminal I/O tests if the special two-asynchronous-port configuration has passed the tests.

A more limited method that may be used if only one asynchronous port is available is to wire the port to loop the output of the port back to the input of the same port.

A PCTS.1 may determine the attributes of an asynchronous port through the use of the *tcsetattr*() and *tcgetattr*() functions. An attribute, such as CLOCAL for modem control, may be set using *tcsetattr*(). If that value is not set after being set by *tcsetattr*() and read by *tcgetattr*(), then modem control is not supported by that serial port.

1.4.16 Test Methods for File Formats

1.4.16.1 Requirements

The PCTS.1 shall supply all files used to test file formats (see Section 10.). These file formats are tested using the file reading utility. Do not create them on the target system with the file creating utility.

Where the PCTS.1 tests the content of text in file format headers, it shall do so using the ISO/IEC 646 :1991 {4} IRV character representation (see Section 10.).

1.4.16.2 Recommendations

None.

1.4.17 Documentation Audit

1.4.17.1 Requirements

The purpose of the documentation audit is to verify that the PCD.1 meets the requirements specified in Section 1.3.1.2 of POSIX.1 {3} as enumerated by the documentation assertions. These general requirements are that the PCD.1

- Has the same structure as POSIX.1 {3}, with the information presented in the appropriately numbered sections, clauses, and subclauses. Information may appear under a higher level heading or in a section, clause, or subclause whose content in POSIX.1 {3} is intended to cover multiple sections, clauses, or subclauses throughout the standard.
- Does not contain information about extended facilities or capabilities outside the scope of POSIX.1 {3} and only contains information related to the statements in Section 1.3.1.2 of POSIX.1 {3}
- Describes the limit values found in the <limits.h> and <unistd.h> headers and listed in Sections 2.8 and 2.9 of POSIX.1 {3}, stating values, the conditions under which those values may change, and the limits of such variations, if any
- Describes the behavior of the implementation for all implementation-defined features defined in POSIX.1 {3}
- May specify the behavior of the implementation for those features where POSIX.1 {3} states that implementations may vary or implies variations among implementations
- May specify the behavior of the implementation for those features where features are identified as undefined or unspecified, or where the POSIX.1 {3} text implies these terms
- Specifies all items of POSIX.1 {3} where the phrases "shall document" or "shall be documented" are used

Extended security controls may necessitate features to be documented that do not have explicit documentation assertions. Such additional documentation shall be allowed as long as it is revelant to features specified in POSIX.1 {3}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The test result code appropriate for each documentation assertion shall be determined. When the information specified by a documentation assertion does not appear in the designated section or sections, the result code for that assertion shall be **FAIL**.

Conformance documents, required by standard organizations, may be bound together as long as the POSIX.1 {3} part is clearly identifiable.

1.4.17.2 Recommendations

None.

1.4.18 PCD Announcement Mechanism

1.4.18.1 Requirements

When assertions are dependent on the PCD.1 for an announcement mechanism about the support or nonsupport of POSIX.1 {3} conditional features, the symbol to be used and its associated conditional feature are stated in Table 1.4.

Table 1.4—PCD Determinable Conditional Features

Symbol	Documentation
{PCD_CREAT_LINK_COUNT}	If supported, then creating a directory causes the link count of the directory in which it is created to be incremented.
{PCD_DIR_TYPE}	If supported, then type DIR is implemented as a file descriptor.
{PCD_LINK_TO_DIRECTORY}	If supported, then a calling process with the proper permission can create a link to a directory.
{PCD_LINK_FILE_SYSTEM}	If supported, then a link can be created between different file systems.
{PCD_NO_LOCK_FILE_TYPE}	If supported, then file types for which locking is unsupported do exist.
{PCD_READ_INTERRUPTED}	If supported, then *read*() when interupted by a signal after successfully reading some data returns the number of bytes read, otherwise returns an error value.
{PCD_WRITE_INTERRUPTED}	If supported, then *write*() when interupted by a. signal after successfully writing some data returns number of bytes written, otherwise returns an error value.
{PCD_WRITE_PERM_TO_RENAME}	If supported, then write permission is required for an existing directory to rename it.

1.4.18.2 Recommendations

None.

1.4.19 Test Methods for Common-Usage C Functions

1.4.19.1 Requirements

If an implementation claiming conformance to Common-Usage C does not support a C Standard {2} function or a PCTS.1 detects a deviation from the C Standard {2}, the corresponding assertions in 8.1.1 through 8.1.59 shall have

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

the result code **UNRESOLVED** or **FAIL**. If the audit of the PCD.1 shows that the PCD.1 declares the C function not implemented or the deviations are documented, the test result code is resolved to **PASS**; otherwise it remains **FAIL**. Additionally, if differences are not documented in the PCD.1, documentation assertion "DO1" in Section 8. shall have the result code **FAIL**.

This means that effectively each assertion in 8.1.1 through 8.1.59 (although not written that way) reads as follows:

XX(A) If the function is supported and there is no deviation from (assertion):
(assertion)

Otherwise:
Either the PCD.1 declares the C function not implemented, or the PCD.1 documents the deviation.

1.4.19.2 Recommendations

None.

2. Definitions and General Requirements

2.1 Conventions

This standard uses the following typographic conventions:

1) IEEE Std 1003.3-1991 {4} defines the nomenclature required for writing assertions. The description of the terms (A), (B), (C), (D), (Condition?A:B), R, GA, and the testing requirements are discussed in that standard.
2) When assertions are dependent on testing constraints, the nomenclature as described in 1.4.3.3 is used.
3) When assertions are dependent on PCD conditionals, the nomenclature as described in 1.4.3.4 is used.
4) Assertion lists generated for each POSIX.1 {3} element begin with a new clause or subclause. Each clause and subsequent subclauses contain a trailing POSIX.1 {3} subclause reference in parentheses.
5) The *italic* font is used for
   — The initial appearances of defined terms
   — Cross-references to defined terms within 1.4.2
   — Parameters (option arguments and operands) that are generally substituted with real values by the application
   — C-language data types and function names
   — Global external variable names
   — The trailing POSIX.1 {3} subclause reference of the element
6) The **bold** font is used for
   — Test result codes
7) The `constant-width` (Courier) font is used
   — To illustrate examples of system input or output where exact usage is depicted
   — For references to utility names, element groupings, and C-language headers
8) Symbolic *errno* names returned by functions are represented as [Symbolic_name].
9) Symbolic constant limits are represented as {Symbolic_constant_limit}.
10) Symbolic constant options are represented as {Option_name}.
11) Notes provided as parts of labeled tables and figures are integral parts of this standard (normative). Footnotes and notes within the body of the text are for information only (informative).
12) Defined names that are usually in lowercase, particularly function names, are never used at the beginning of a sentence or anywhere else that regular English usage would require them to be capitalized.
13) Mathematical symbols such as <, ≤, etc. are only used in formulas, assertion classification assignments, and conditional clauses preceding conditional assertions.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

14) In some cases tabular information is presented "inline;" in others it is presented in a separately labeled table. This arrangement was employed purely for ease of typesetting, and there is no normative difference between these two cases.
15) The double quote ("name") is used when a name is specified.
16) The single quote ('value') is used when a specific value is specified.
17) The conventions listed above are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in this standard.

A summary of typographical conventions is shown in Table 2.1.

Table 2.1—Typographical Conventions

Reference	Example
C language header	`<sys/stat.h>`
Command name	`tar`
Data types	*long*
Defined terms	*file*
Environment variables	**PATH**
Error number	[EINTR]
File name	`filename`
Function argument declaration	extern unsigned long int
Function argument	*arg1*
Function declaration	long int
Function name	*funct*()
Global external	*errno*
IEEE Std 1003 .n-19xy reference	POSIX.n
Implementation-dependent limit	{MAX_INPUT}
Metacharacter	*(any character string)
Null test method	UNUSED
Parameters	*<path>*
Section x.y reference	See x.y or x.y
Special character	`<newline>`
Symbolic constant limit	{LINK_MAX}
Symbolic constant option	{_POSIX_JOB_CONTROL}
Symbolic constant value	{_POSIX_JOB_CONTROL}
Symbol defined in header	_POSIX_JOB_CONTROL
Table reference	Table 2.1
Variable	*st_atime*

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.2 Definitions

2.2.1 Terminology

There are no test methods provided for this subclause.

2.2.2 General Terms

NOTE — The subclause numbers that follow are not sequential because not all definitions have associated test assertions. Only those with specific test assertions are presented here.

2.2.2.4 Appropriate Privileges

D01(A) The means for associating the appropriate privileges with a process are described in 2.2.2.4 of the PCD.1 or in all the subclauses of the PCD.1 that correspond to subclauses in POSIX.1 {3} where privileged behavior is described.

2.2.2.9 Character Special File

D02(C) If character special files other than terminal device files are supported, and this is documented:
The details are contained in 2.2.2.9 of the PCD.1.

D03(C) If structures of character special files other than terminal device files are described:
The details are contained in 2.2.2.9 of the PCD.1.

2.2.2.27 File

D04(C) If file types other than regular file, character special file, block special file, FIFO special file, and directory are described:
The details are contained in 2.2.2.27 of the PCD.1.

2.2.2.30 File Group Class

D05(C) If the implementation uses additional criteria beyond those specified by POSIX.1 {3} to assign a process to the file group class of a file, and this is documented:
The details are contained in 2.2.2.30 of the PCD.1.

2.2.2.32 Filename

GA02 For *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing, link*() *new, mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old, rename*() *new, star*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), *remove*(), `tar` format creating utility, and `cpio` format creating utility:
An implementation supports *filenames* containing any of the following characters:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

GA03 For *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing, link*() *new, mkdir*(), *inkfifo*(), *unlink*(), *rmdir*(), *rename*() *old, rename*() *new, stat*(), *access*(), *chmod*(), *chown*(), *utirne*(), *pathconf*(), *fopen*(), *freopen*(), *remove*(), `tar` format creating utility, and `cpio` format creating utility:
A *filename* retains the unique identity of its upper- and lowercase letters.

01(B) No two directory entries in the same directory have the some name.

See Reason 3 in Section 5. of POSIX.3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.2.2.55 Parent Process ID

D06(A) The new parent process ID assigned to a process after the lifetime of its parent has ended is stated in 2.2.2.55 of the PCD.1.

2.2.2.57 Pathname

D07(A) Any special interpretation of a *pathname* that begins with exactly two slashes is stated in 2.2.2.57 of the PCD.1.

2.2.2.64 Process Group ID

02(B) A process group ID is not reused by the system until the process group lifetime ends.

See Reason 3 in Section 5. of POSIX. 3 [4].

2.2.2.66 Process Group Lifetime

03(A) The lifetime of the process group ends when the last remaining process leaves the group.

Testing Requirements:

Test that the process group for the last remaining process in the process group that has been terminated but unwaited for still exists.

2.2.2.67 Process ID

04(B) A process that is not a system process does not have a process ID of 1.

See Reason 3 in Section 5. of POSIX.3 [4].

05(B) A process ID is not reused by the system until the process lifetime ends.

See Reason 3 in Section 5. of POSIX.3 [4].

06(B) A process ID is not reused while there exists a process group with a process group ID of the same number.

See Reason 3 in Section 5. of POSIX. 3 [4].

2.2.2.68 Process Lifetime

D08(C) If documentation is provided that states the resources returned when a process terminates:

The details are contained in 2.2.2.68 of the PCD.1.

07(B) When a process completes a *wait*() or *waitpid*() for a process that is inactive due to termination, then all of the resources associated with the inactive process are returned to the system. The last of the resources returned is the process ID.

See Reason 3 in Section 5. of POSIX. 3 [4].

2.2.2.69 Read-Only File System

D09(A) The specific restrictions on the semantics of system interfaces as they apply to objects on read-only file systems are contained in 2.2.2.69 of the PCD.1.

2.2.2.83 Supplementary Group ID

D10(C) If the effective group ID of a process is included in the list of supplementary group IDs returned by *getgroups*(), and this is documented:

The details are contained in 2.2.2.83 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.3 General Concepts

2.3.1 Extended Security Controls

D01(C) If extended security controls are supported, and this is documented:
The extended security controls supported are stated in 2.3.1 of the PCD.1.

01(D) If the implementation supports extended security controls:
Extended security controls do not alter or override the defined semantics of any of the functions in POSIX.1 {3}.

See Reason 1 in Section 5. of POSIX.3 {4}.

2.3.2 File Access Permissions

D02(C) If whether additional or alternate file access control mechanisms are supported is documented:
The manner of support for additional or alternate file access control mechanisms or information that they are not supported is contained in 2.3.2 of the PCD.1.

GA04 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), and *pathconf*():
If the implementation provides the mechanism for creating processes with the appropriate privilege to override a file access control mechanism:
When the process has appropriate privileges for file access, and a call to function *funct*() requires read, write, or search access to the *pathname* argument, then the process is granted access to the file.

GA05 For *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), and *execvp*():
If the implementation provides the mechanism for creating processes with the appropriate privilege to override a file access control mechanism:
When the process has the appropriate privileges for file access, execute permission is granted to at least one user of the file, and a call to an `exec` type function requires execute access to the argument *path* or *file*, then the process is granted access.

GA06 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), and *pathconf*():
When the process does not have the required appropriate privileges to override the file access control mechanism, and the process requires read, write, execute, or search access to the pathname argument of function *funct*(), then a call to *funct*() is granted access in the case that the required access permission bit is set for the class (file owner class, file group class, or file other class) to which the process belongs.

GA07 If the implementation supports additional file access control mechanisms:
The mechanism only further restricts the access permissions defined by the file permission bits.

GA08 If the implementation supports alternate file access control mechanisms:
The mechanism specifies file permission bits for the file owner class, file group class, and file other class of the file corresponding to the access permissions to be returned by *stat*() and *fstat*().

GA09 If the implementation supports alternate file access control mechanisms:
The mechanism can be enabled only by explicit user action on a per-file basis by a process having the required appropriate privilege.

GA10 If the implementation supports alternate file access control mechanisms:
The mechanism can be enabled only by explicit user action on a per-file basis by the file owner.

GA11 If the implementation supports alternate file access control mechanisms:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The mechanism can be disabled for a file after the file permission bits are changed for that file with *chmod*().

D03(C) If both alternate and additional file access control mechanism(s) are provided, and whether disabling an alternate mechanism after *chmod*() disables additional mechanisms is documented:
The details are contained in 2.3.2 of the PCD.1.

2.3.3 File Hierarchy

There are no test methods provided for this subclause.

2.3.4 Filename Portability

There are no test methods provided for this subclause.

2.3.5 File Times Update

R01 For *close*():
When a file is closed by the last process that had it open, then all time-related fields marked for update are updated. [See Assertion 11 in 6.3.1.2.]

GA12 For *stat*() and *fstat*():
When *stat*() or *fstat*() is called for a file, then all time-related fields marked for update are updated, and time-related fields not marked for update are not updated.

D04(C) If an implementation of a function that is required by POSIX.1 {3} to update time-related fields for reasons other than pathname resolution updates fields other than those required:
The details are contained in 2.3.5 of the PCD.1 or in the subclause of the PCD.1 that corresponds to the *description* subclause in POSIX.1 {3} where the function is defined.

D05(C) If an implementation of a function that is not required by POSIX.1 {3} to update time-related fields happens to do so, and this is documented:
The details are contained in 2.3.5 of the PCD.1 or in the subclause of the PCD.1 that corresponds to the *description* subclause in POSIX.1 {3} where the function is defined.

D06(C) If whether the updating of the related fields is done immediately or periodically is documented:
The details are contained in 2.3.5 of the PCD.1 or in the subclause of the PCD.1 that corresponds to the *description* subclause in POSIX.1 {3} where the function is defined.

GA13 For *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *access*(), *chmod*(), *chown*(), and *utime*():
No time-related fields are updated on read-only file systems.

2.3.6 Pathname Resolution

NOTE — In each of the pathname resolution general assertions below, for the elements *rmdir*(), *rename*() new, and *unlink*(), the current working directory should be an empty directory in order to avoid the occurrence of avoidable error conditions. Some implementations will consider the attempt to remove the current working directory an error and will indicate this with the [EBUSY] error indication.

GA14 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():
When the first filename component of the pathname argument is " . ", and the pathname does not begin with a slash, then *funct*() resolves the pathname by locating the second filename component (when specified) in the current working directory.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860170 35T ■

GA15 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the argument *path* or *file* points to the string "/", then *funct*() resolves the pathname to the root directory of the process.

GA16 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the argument *path* or *file* points to the string "///", then *funct*() resolves the pathname to the root directory of the process.

GA17 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the pathname argument points to a string beginning with a single slash or beginning with three or more slashes, then *funct*() resolves the pathname by locating the first filename component of the pathnome in the root directory of the process.

GA18 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *exedp*(), *execvp*(), *opendir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the first filename component of the pathname argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *funct*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory.

GA19 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the argument *path* or *file* points to the string "`F1/`" and `F1` is a directory, then *funct*() resolves the pathname to `F1`, which is in the current working directory.

GA20 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the argument *path* or *file* points to the string "`F1//`" and `F1` is a directory, then *funct*() resolves the pathname to `F1`, which is in the current working directory.

For *rename*() new:

When the argument *new* points to the string "`F1//`" and `F1` is an empty directory, then *rename*() resolves the pathname to `F1`, which is in the current working directory.

GA21 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *star*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the pathname argument points to the string "`F1/F2`", then *funct*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current, working directory.

GA22 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the pathname argument points to the string "`F1/./F2`", then *funct*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory.

GA23 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rrndir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(),*chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the pathname argument points to the string "F1/../F1/F2", then *funct*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory.

GA24 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

When the pathname argument points to the string "`F1//F2`", then *funct*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory.

GA25 For *funct*() of *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *open-dir*(), *chdir*(), *open*(), *creat*(), *link*() *existing*, *link*() *new*, *mkdir*(), *mkfifo*(), *unlink*(), *rmdir*(), *rename*() *old*, *rename*() *new*, *stat*(), *access*(), *chmod*(), *chown*(), *utime*(), *pathconf*(), *fopen*(), *freopen*(), and *remove*():

If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

When the pathname component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC}; is not supported, then *funct*() resolves the pathname component by truncating it to {NAME_MAX} bytes.

There are no assertions specific to this subclause.

2.4 Error Numbers

01(A) The element *errno* is defined and available to a process via

extern int errno;

02(A) When the header `<errno.h>` is included, then the error numbers [E2BIG], [EACCES], [EAGAIN], [EBADF], [EBUSY], [ECHILD], [EDEADLK], [EDOM], [EEXIST], [EFAULT], [EFBIG], [EINTR], [EINVAL], [EIO], [EISDIR], [EMFILE], [EMLINK], [ENAMETOOLONG], [ENFILE], [ENODEV], [ENOENT], [ENOEXEC], [ENOLCK], [ENOMEM], [ENOSPC], [ENOSYS], [ENOTDIR], [ENOTEMPTY], [ENOTTY], [ENXIO], [EPERM], [EPIPE], [ERANGE], [EROFS], [ESPIPE], [ESRCH], and [EXDEV] are defined, are nonzero, are distinct from each other, and can be represented in *errno*.

03(B) No function defined by POSIX.1 sets *errno* to zero to indicate an error.

See Reason 2 in Section 5. of POSIX. 3 [4].

D01(C) If the implementation supports additional errors not listed in 2.4, and this is documented:

The details are contained in 2.4 of the PCD.1 or in the subclause of the PCD.1 that corresponds to the errors subclause in POSIX.1 {3} for each interface that returns the error.

DGA02 For *fork*(), *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*(), *sigaddset*(), *sigdelset*(), *sigismernber*(), *opendir*(), *readdir*(), *closedir*(), *getcwd*(), *access*(), *chown*(), *pathconf*(), *fpathconf*(), and *fcntl*():

If the implementation supports the detection of optional error conditions:

The details of the error conditions detected are contained in the subclause of the PCD.1 where the error values of the function are described.

GA26 For *fork*(), *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*(), *sigaddset*(), *sigdelset*(), *sigismernber*(), *opendir*(), *readdir*(), *closedir*(), *access*(), *chown*(), and *fcntl*():

If the implementation does not support the detection of optional error conditions:

The associated action requested is successful (unless a different error condition is detected).

D02(A) The details concerning whether [EFAULT] is reliably detected are contained in 2.4 of the PCD.1 or in the subclause of the PCD.1 that corresponds to the *errors* subclause in POSIX.1 {3} for each interface that is documented optionally to return [EFAULT].

D03(A) The maximum file size allowed by the implementation is contained in 2.4 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.5 Primitive System Data Types

01(A) When the header <sys/types.h> is included, then the data types *dev_t, gid_t, ino_t, mode_t, nlink_t, off_t, pid_t, size_t, ssize_t,* and *uid_t* are defined and are arithmetic data types.

02(A) The data type of each of *pid_t, off_t,* and *ssize_t* is signed arithmetic.

03(A) The data type of *size_t* is unsigned arithmetic.

04(A) The type *ssize_t* is capable of storing values in a range from "−1" to SSIZE_MAX.

05(B) When the header <unistd.h> is included, then the data types *size_t* and *ssize_t* are defined.

See Reason 1 in Section 5. of POSIX. 3 {4}.

D01(C) If additional type symbols ending in "_t" are defined in any header specified by POSIX.1 {3}, and this is documented:

The details are contained in 2.5 of the PCD.1.

2.6 Environment Description

01(C) If the environment variable **HOME** was defined by the implementation and currently has the value defined by the implementation:

The environment variable **HOME** corresponds to the *initial working directory* of the user from the user database.

02(D) If the environment variable **LOGNAME** was defined by the implementation and currently has the value defined by the implementation:

The environment variable **LOGNAME** corresponds to the *login name* associated with the current process.

NOTE — Testing that **LOGNAME** is composed of *characters from the portable filename character set* is not asserted since POSIX.1 {3} implies in a note that this condition should be tolerated.

See Reason 2 in Section 5. of POSIX.3 {4}.

GA27 For *execlp*() and *execvp*():

Only when the argument *file* does not contain a slash are the **PATH** prefixes used by *execlp*() and *execvp*().

GA28 For *execlp*() and *execvp*():

A colon ':' in **PATH** separates one path prefix from another. Each path prefix in **PATH** indicates a directory to be included in the search path.

GA29 For *execlp*() and *execvp*():

When a nonzero-length prefix is applied to a filename, then a '/' is inserted between the prefix and the filename.

GA30 For *execlp*() and *execvp*():

An occurrence of two adjacent colons '::' in **PATH** indicates that the current working directory is to be included in the search path.

GA31 For *execlp*() and *execvp*():

An initial ':' indicates the current working directory.

GA32 For *execlp*() and *execvp*():

A trailing ':' indicates the current working directory.

GA33 For *execlp*() and *execvp*():

The directories in the search path are searched in the order in which they occur in the **PATH** environment until an executable file of the specified name is found.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

GA34 For *getenv*() and exec type functions:
Upper- and lowercase letters in the environment retain their unique identities.

GA35 For *getenv*() and exec type functions:
An implementation supports environment variable names consisting of characters in the portable filename character set.

D01(C) If the implementation supports the use of characters other than those in the portable character set in environment variable names, and this is documented:
The details are contained in 2.6 of the PCD.1.

2.7 C Language Definitions

2.7.1 Symbols From the C Standard

01(A) When the header <time.h> is included, then the types *clock_t* and *time_t* and the macro **NULL** are defined.

02(A) The type *clock_t* is capable of representing all integer values from zero to the number of clock ticks in 24 h.

03(A) When the header <unistd.h> is included, then the symbol **NULL** is defined and expands to either an integral constant expression with the value 0 or such an expression cast to type *(void *)*.

04(A) Each of the following headers <dirent.h>, <fcntl.h>, <grp.h>, <pwd.h>, <setjmp.h>, <signal.h>, <stdio.h>, <sys/stat.h>, <sys/times.h>, <sys/wait.h>, <termios.h>, <time.h>, <utime.h>, and <unistd.h> can be included more than once, in any combination of headers in any order, and a symbol can be defined with the same value in more than one of these headers.

NOTE — It is possible to include <sys/types.h> more than once, and the first instance of its inclusion precedes any other header that depends upon its prior inclusion.

2.7.2 POSIX.1 Symbols

NOTE — When {_POSIX_SOURCE} is defined before any header is included, then those symbols defined by POSIX.1 {3}; to be in the header are visible. (See 1.4.7.1.)

05(B) When {_POSIX_SOURCE} is defined and no other feature test macros are defined before any header is included, then no symbols other than those defined or reserved for that header by POSIX.1 {3} or the C Standard {2} are made visible. The constraints on the usage of reserved symbols are as specified in POSIX.1 {3}.

See Reason 5 in Section 5. of POSIX. 3 {4}.

D01(C) If the implementation supports feature test macros other than {_POSIX_SOURCE}, and this is documented:
The details are contained in 2.7.2 of the PCD.1.

2.7.2.1 C Standard Language-Dependent Support

There are no assertions specific to this subclause.

2.7.2.2 Common-Usage-Dependent Support

There are no assertions specific to this subclause.

2.7.3 Headers and Function Prototypes

GA36 For all elements with result type not *void* and not *int:*
If the implementation provides C Standard {2} support:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the header <*.h> is included, then the function prototype *type1 funct(type2, type3)* is declared.[5]

Otherwise:

When the header <*.h> is included, then the function *funct*() is declared with the result type *type1*.

For all elements with result type *void* except *assert*():

If the implementation provides C Standard {2} support:

When the header <*.h> is included, then the function prototype *void funct(type2, type3)* is declared.

For all elements with result type *int except setjmp*() and *sigsetjmp*():

If the implementation provides C Standard {2} support:

When the header <*.h> is included, then the function prototype *int funct(type2, type3)* is declared.

Otherwise:

When the header <*.h> is included, then the function *funct*() is either declared with the result type *int* or is not declared in the header.

For *funct*() of *setjmp*() and *sigsetjmp*():

If *funct*() is not defined as a macro:

When the header <setjmp.h> is included, then *funct*() is declared as an identifier with external linkage and result type *int*.

GA37 If *funct*() is defined as a macro when the header <*.h> is included:

When the macro *funct*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), it then expands to an expression with the result type compatible with that specified for *funct*().

There are no assertions specific to this subclause.

2.8 Numerical Limits

2.8.1 C Language Limits

01(A) When the header <limits.h> is included, then the following symbols and corresponding values are defined and have values that meet the requirements of the C Standard {2}, as shown in Table 2.2.

[5]The function prototypes are aligned with ISO/IEC 9945-1 : 1990. When not specified in ISO/IEC 9945-1 : 1990, they are aligned with ISO/IEC 9899 :1990.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Table 2.2—C Language Limits

Symbol	Value	Description
CHAR_BIT	8	Minimum number of bits
CHAR_MAX		See note below
CHAR_MIN		See note below
INT_MAX	+32767	Minimum maximum
INT_MIN	-32767	Maximum minimum
LONG_MAX	+2147483647	Minimum maximum
LONG_MIN	-2147483647	Maximum maximum
MB_LEN_MAX	1	Minimum maximum
SCHAR_MAX	+127	Minimum maximum
SCHAR_MIN	-127	Maximum minimum
SHRT_MAX	+32767	Minimum maximum
SHRT_MIN	-32767	Maximum minimum
UCHAR_MAX	255	Minimum maximum
UINT_MAX	65535	Minimum maximum
ULONG_MAX	4294967295	Minimum maximum
USHRT_MAX	65535	Minimum maximum

NOTE — If *char* is signed, then {CHAR_MIN} is equal to {SCHAR_MIN} and {CHAR_MAX} is equal to {SCHAR_MAX}; otherwise, {CHARMIN} is zero and {CHAR_MAX} is equal to {UCHAR_MAX}.

2.8.2 Minimum Values

02(A) When the header <limits.h> is included, then the following symbols and corresponding values are defined as shown in Table 2.3.

Table 2.3—Minimum Values

Symbol	Value
{_POSIX_ARG_MAX}	4096
{_POSIX_CHILD_MAX}	6
{_POSIX_LINK_MAX}	8
{_POSIX_MAX_CANON}	255
{_POSIX_MAX_INPUT}	255
{_POSIX_NAME_MAX}	14
{_POSIX_NGROUPS_MAX}	0
{_POSIX_OPEN_MAX}	16
{_POSIX_PATH_MAX}	255
{_POSIX_PIPE_BUF}	512
{_POSIX_SSIZE_MAX}	32767
{_POSIX_STREAM_MAX}	8
{_POSIX_TZNAME_MAX}	3

R01 A conforming implementation provides values at least as large as those specified in the previous table for the related symbols of the conforming implementation. (See Assertions 4–15 in 2.8.4 and 2.8.5.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.8.3 Run-Time Increasable Values

D01(A) The magnitude limitation fixed for {NGROUPS_MAX} (defined in <limits.h>) is described in 2.8.3 of the PCD.1.

03(A) When the header <limits.h> is included, then the symbolic name {NGROUPS_MAX} is defined, and the value of the symbolic constant is not less than the value of {_POSIX_NGROUPS_MAX}.

2.8.4 Run-Time Invariant Values

04(C) If the symbolic constant {ARG_MAX} is defined:
When the header <limits.h> is included, then the value of {ARG_MAX} is not less than {_POSIX_ARG_MAX}.

05(C) If the symbolic constant {CHILD_MAX} is defined:
When the header <limits.h> is included, then the value of {CHILD_MAX} is not less than {_POSIX_CHILD_MAX}.

06(C) If the symbolic constant {OPEN_MAX} is defined:
When the header <limits.h> is included, then the value of {OPEN_MAX} is not less than {_POSIX_OPEN_MAX}.

07(C) If the symbolic constant {STREAM_MAX} is defined:
When the header <limits.h> is included, then the value of {STREAM_MAX} is not less than {_POSIX_STREAM_MAX}.

08(C) If the symbolic constant {STREAM_MAX} is defined:
When the header <limits.h> is included, then the value of {STREAM_MAX} is equal to the value of FOPEN_MAX in the header <stdio.h>.

09(C) If the symbolic constant {TZNAME_MAX} is defined:
When the header <limits.h> is included, then the value of {TZNAME_MAX} is not less than {_POSIX_TZNAME_MAX}.

D02(A) The run-time invariant values for {ARG_MAX}, {CHILD_MAX}, {OPEN_MAX}, {STREAM_MAX}, and {TZNAME_MAX} (defined in <limits.h>), are contained in 2.8.4 of the PCD.1.

2.8.5 Pathname Variable Values

10(C) If the symbolic constant {LINK_MAX} is defined:
When the header <limits.h> is included, then the value of {LINK_MAX} is not less than {_POSIX_LINK_MAX}.

11(C) If the symbolic constant {MAX_CANON} is defined:
When the header is included, then the value of {MAX_CANON} is not less than {_POSIX_MAX_CANON}.

12(C) If the symbolic constant {MAX_INPUT} is defined:
When the header <limits.h> is included, then the value of {MAX_INPUT} is not less than {_POSIX_MAX_INPUT}.

13(C) If the symbolic constant {NAME_MAX} is defined:
When the header <limits.h> is included, then the value of {NAME_MAX} is not less than {_POSIX NAME_MAX}.

14(C) If the symbolic constant {PATH_MAX} is defined:
When the header <limits.h> is included, then the value of {PATH_MAX} is not less than {_POSIX_PATH_MAX}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15(C) If the symbolic constant {PIPE_BUF} is defined:
When the header <limits.h> is included, then the value of {PIPE_BUF} is not less than {_POSIX_PIPE_BUF}.

D03(A) The values for {LINK_MAX}, {MAX_CANON}, {MAX_INPUT}, {NAME_MAX}, {PATH_MAX}, and {PIPE_BUF} (defined in <limits.h>), conditions under which these values may change, and the limits of such variations, if any, are contained in 2.8.5 of the PCD.1.

2.8.6 Invariant Values

16(D) If the implementation provides a means of obtaining the value of {SSIZE_MAX} at runtime:
The runtime value of {SSIZE_MAX} is equal to the value of SSIZE_MAX defined in <limits.h>.

See Reason 3 in Section 5. of POSIX.3 [4].

17(A) When the header <limits.h> is included, then the macro SSIZE_MAX is defined and evaluates to a value not less than {_PCSIX_SSIZE_MAX}.

2.9 Symbolic Constants

R01 The implementation has the header <unistd.h>. (See Assertions 1, 3, 5, and 6 in 2.9.1 through 2.9.4.)

D01(A) For constants {_POSIX_JOB_CONTROL} and {_POSIX_SAVED_IDS} (defined in <unistd.h>), the values, conditions under which these values may change, and the limits of such variations, if any, are contained in 2.9 of the PCD.1.

D02(C) If the constant {_POSIX_CHOWN_RESTRICTED} is defined in <unistd.h>:
The value of {_POSIX_CHOWN_RESTRICTED}, conditions under which this value may change, and the limits of such variations, if any, are contained in 2.9 of the PCD.1.

D03(C) If the constant {_POSIX_NO_TRUNC} is defined in <unistd.h>:
The value of {_POSIX_NO_TRUNC}, conditions under which this value may change, and the limits of such variations, if any, are contained in 2.9 of the PCD.1.

D04(C) If the constant {_POSIX_VDISABLE} is defined in <unistd.h>:
The value of {_POSIX_VDISABLE}, conditions under which this value may change, and the limits of such variations, if any, are contained in 2.9 of the PCD.1.

2.9.1 Symbolic Constants for the *access*() Function

01(A) When the header <unistd.h> is included, then the constants R_OK (test for read permission), W_OK (test for write permission), X_OK (test for execute or search permission), and F_OK (test for existence of file) are defined.

02(A) The constants F_OK, R_OK, W_OK, X_OK, and the expression R_OK|W_OK, R_OK|X_OK, W_OK|X_OK, and R_OK|W_OK|W_OK all have distinct integral values.

2.9.2 Symbolic Constants for the *lseek*() Function

03(A) When the header <unistd.h> is included, then the elements SEEK_SET, SEEK_CUR, and SEEK_END are defined.

04(A) The elements SEEK_SET, SEEK_CUR, and SEEK_END all have distinct integral values.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.9.3 Compile-Time Symbolic Constants for Portability Specifications

R02 When the header <unistd.h> is included, then the value of the symbol {_POSIX_JOB_CONTROL} is not less restrictive than that provided by the respective POSIX option via *sysconf*(). (See Assertion 12 in 4.8.1.2.)

R03 When the header <unistd.h> is included, then the value of the symbol {_POSIX_SAVED_IDS} is not less restrictive than that provided by the respective POSIX option via *sysconf*(). (See Assertion 13 in 4.8.1.2.)

05(A) When the header <unistd.h> is included, then {_POSIX_VERSION} is defined and matches the integer value of POSIX.1 {3}, "199009L."

2.9.4 Execution-Time Symbolic Constants for Portability Specifications

D05(C) If {_POSIX_CHOWN_RESTRICTED} is not defined or defined with a value other than −1 in <unistd.h>:

The files and file types to which{_POSIX_CHOWN_RESTRICTED} applies are contained in 2.9.4 of the PCD.1.

R04 When the header <unistd.h> is included, and the value of the symbol {_POSIX_CHOWN_RESTRICTED} is defined and has a value other than −1, then the value of the symbol provides the same indication as that provided by the interrogation at run-time of the POSEC.1 {3} option via *pathconf*() and *fpathconf*(). (See Assertions 36 and 20 in 5.7.1.1.2 and 5.7.1.2.2.)

D06(C) If {_POSIX NO_TRUNC} is not defined or defined with a value other than −1 in <unistd.h>:

The files and file types to which {_POSIX_NO_TRUNC} applies are contained in 2.9.4 of the PCD.1.

R05 When the header <unistd.h> is included, and the value of the symbol {_POSIX_NO_TRUNC} is defined and has a value other than −1, then the value of the symbol provides the same indication as that provided by the interrogation at run-time of the POSIX.1 {3} option via *pathconf*() and *fpathconf*(). (See Assertions 38 and 22 in 5.7.1.1.2 and 5.7.1.2.2.)

D07(C) If {_POSIX_VDISABLE} is not defined or defined with value other than −1 in <unistd.h>:

The files and file types to which {_POSIX_VDISABLE} applies are contained in 2.9.4 of the PCD.1.

R06 When the header <unistd.h> is included, and the value of the symbol {_POSIX_VDISABLE} is defined and has a value other than −1, then the value of the symbol provides the same indication as that provided by the interrogation at run-time of the POSIX.1 {3} option via *pathconf*() and *fpathconf*(). (See Assertions 40 and 24 in 5.7.1.1.2 and 5.7.1.2.2.)

06(D) If {_POSIX_CHOWN_RESTRICTED} is defined with the value −1 when <unistd.h> is included:

When the header <unistd.h> is included and {_POSIX_CHOWN_RESTRICTED} is −1, then the implementation does not provide this option.

See Reason 3 in Section 5. of POSIX.3 {4}.

07(D) If {_POSIX_NO_TRUNC} is defined with the value −1 when <unistd.h> is included:

When the header <unistd.h> is included and {_POSIX_NO_TRUNC} is −1, then the implementation does not provide this option.

See Reason 3 in Section 5. of POSIX.3 {4}.

08(D) If {_POSIX_VDISABLE} is defined with the value −1 when <unistd.h> is included:

When the header <unistd.h> is included and {_POSIX_VDISABLE} is −1, then the implementation does not provide this option.

See Reason 3 in Section 5. of POSIX. 3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3. Process Primitives

3.1 Process Creation and Execution

3.1.1 *fork*()

3.1.1.1 Synopsis

01(A) If the implementation provides C Standard {2}; support:
When the header <unistd.h> is included, then the function prototype `pid_t fork(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *fork*() is declared with the result type *pid_t*. (See GA36 in 2.7.3.)

02(C) If *fork*() is defined as a macro when the header <unistd.h> is included:
When the macro *fork*() is invoked, then it expands to an expression with the result type *pid_t*. (See GA37 in 2.7.3.)

03(C) If *fork*() is defined as a macro in the header <unistd.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.1.1.2 Description

04(A) When a call to *fork*() completes successfully, then the child process ID is different from the parent process ID.

05(B) When a call to *fork*() completes successfully, then the child process ID does not match any other active process ID.

See Reason 1 in Section 5. of POSIX.3 {4}.

06(B) When a call to *fork*() completes successfully, then the child process ID does not match any active process group ID.

See Reason 1 in Section 5. of POSIX.3 {4}.

07(A) When a call to *fork*() completes successfully, then the parent process ID of the child process is the process ID of the parent process.

08(A) When a call to *fork*() completes successfully, then each of the file descriptors for the child refers to the same open file description as the corresponding file descriptor of the parent.

09(A) When a call to *fork*() completes successfully, then the child process has its own copy of the open directory streams of the parent process.

10(A) When a call to *fork*() completes successfully, then the values of *tms_utime, tms_stime, tms_cutime,* and *tms_cstime* for the child process are reset.

D01(C) If the implementation supports sharing open directory streams between parent and child processes, and this is documented:
The details are contained in 3.1.1.2 of the PCD.1.

11(A) When a call to *fork*() completes successfully, then file locks set by the parent are not inherited by the child.

12(A) When a call to *fork*() completes successfully, then pending alarms are cleared for the child process.

13(A) When a call to *fork*() completes successfully, then the set of signals pending for the child process is initialized to the empty set.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

14(A) When a call to *fork*() completes successfully, then the current working directory and the root directory are the same as for the parent.

15(A) When a call to *fork*() completes successfully, then signal settings for the child process are the same as for the parent.

16(A) When a call to *fork*() completes successfully, then the real and effective user ID and group ID are the same as for the parent.

17(C) If the behavior associated with {_POSIX_SAVED_IDS} is supported:
When a call to *fork*() completes successfully, then the saved set-user ID and saved set-group ID are the same as for the parent.

18(A) When a call to *fork*() completes successfully, then the supplementary groups for the child are the same as for the parent.

19(A) When a call to *fork*() completes successfully, then the process group for the child is the same as for the parent.

20(PCTS_GTI_DEVICE?A:UNTESTED)

When a call to *fork*() completes successfully, then the controlling terminal for the child process is the same as for the parent.

21(A) When a call to *fork*() completes successfully, then the file mode creation mask is the same as for the parent.

22(A) When a call to *fork*() completes successfully, then the signal mask for the child process, is the same as for the parent.

23({_POSIX_JOB_CONTROL}?A:UNTESTED)

When a call to *fork*() completes successfully, then the child process is in the same session as its parent.

See Reason 1 in Section 5. of POSIX.3 {4}.

D02(C) If the implementation documents whether there is support for inheritance of process characteristics for *fork*() not defined in POSIX.1 {3}:
The details are contained in 3.1.1.2 of the PCD.1

24(A) When a call to *fork*() completes successfully, then both the parent and child process are capable of executing independently before either process terminates.

3.1.1.3 Returns

25(A) When a call to *fork*() completes successfully, then a value of (*pid_t*)0 is returned to the child process, and the child process IE, is returned to the parent process.

R01 When a call to *fork*() completes unsuccessfully, then a value of (*pid_t*)−1 is returned and sets *errno* to indicate the error. (See Assertions 26-28 in 3.1.1.4.)

3.1.1.4 Errors

26(B) When the system lacks the necessary resources to create another process, then a call to *fork*() returns a value of (*pid_t*)-1, sets *errno* to [EAGAIN], and no process is created.

See Reason 3 in Section 5. of POSIX.3 {4}.

27(PCTS_PROCESS_LIMIT?A:UNTESTED)

If {CHILD_MAX} ≤ {PCTS CHILD MAX}:
When the system-imposed limit on the total number of processes under execution by a single user {CHILD_MAX} is reached, then a call to *fork*() returns a value of (*pid_t*)-1, sets *errno* to [EAGAIN], and no process is created.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Otherwise:

{PCTS_CHILD_MAX} processes can be created.

D03(C) If the implementation supports the detection of [ENOMEM] for *fork*():

The details under which [ENOMEM] occurs for *fork*() are contained in 3.1.1.4 of the PCD.1. (See DGA02 in 2.4.)

28(B) If the implementation supports the detection of [ENOMEM] for *fork*():

When the process requires more space than the system is able to supply, then a call to *fork*() returns a value of (*pid_t*) −1, sets *errno* to [ENOMEM], and no process is created.

See Reason 1 in Section 5. of POSIX.3 {4}.

Otherwise:

When the process requires more space than the system is able to supply, then a call to *fork*() is successful (unless a different error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 {4}.

3.1.2 *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), *execvp*()

NOTE — The term *exec* is a notation for each of the six functions; *execl*(), *execle*(), *execv*(), *execve*(), *execlp*(), and *execvp*().

3.1.2.1 Synopsis

01(A) For *execl*():

If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `int execl(const char *, const char *, ...)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *execl*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

For *execv*():

If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `int execv(const char *, char * const [])` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *execv*() is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

For *execle*():

If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `int execle(const char *, const char *,... )` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *execle*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

For *execve*():

If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `int execve(const char *, char * const [], char * const [ ])` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *execve*() is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

For *execlp*():

If the implementation provides C Standard {2} support:

When the header <unistd.h> is included: then the function prototype
`int execlp(const char *, const char *, ...)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *execlp*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

For *execvp*():

If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype
`int execvp(const char *, char * const [])` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *execvp*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) For *execl*():

If *execl*() is defined as a macro when the header <unistd.h> is included:

When the macro *execl*() is invoked with the correct argumenttypes (or compatible argument types in the case, that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

For *execv*():

If *execv*() is defined as a macro when the header <unistd.h> is included:

When the macro *execv*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

For *execle*():

If *execle*() is defined as a macro when the header <unistd.h> is included:

When the macro *execle*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type int. (See GA37 in 2.7.3.)

For *execve*():

If *execve*() is defined as a macro when the header <unistd.h> is included:

When the macro *execve*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

For *execlp*():

If *execlp*() is defined as a macro when the header <unistd.h> is included:

When the macro *execlp*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

For *execvp*():

If *execvp*() is defined as a macro when the header <unistd.h> is included:

When the macro *exevup*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If an `exec` type function is defined as a macro in the header <unistd.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 4.3.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.1.2.2 Description

04(A) There is no return from a successful call to an exec type function, and the calling process image is overlaid by the new process image.

05(A) The *argc* parameter to *main*() of the new process image is the count of the number of argument parameters passed (not including the terminating **NULL** pointer).

06(A) The *argv* parameter to *main*() of the new process image is a null-terminated array of character pointers to the arguments that were provided by the exec type function as the arguments for the new process image.

07(A) For *execle*() and *execve*():
The environment for the new process image is obtained from the argument *envp* (an array of character pointers to null-terminated strings that are terminated by a **NULL** pointer).

For *execl*(), *execv*(), *execlp*(), and *exevup*():
The environment for the new process image is obtained from the external variable *environ* in the calling process image.

08(A) The variable extern char ***environ* in the new process image is initialized as a pointer to an array of character pointers to the environment strings provided by the calling process. The array terminates with a **NULL** pointer.

09(A) For *execlp*() and *execvp*():
PATH prefixes are only used by *execlp*() and *exevup*() when the argument *file* does not contain a '/'. (See GA27 in 2.6.)

For *execl*(), *execv*(), *execle*(), and *execve*():
UNUSED.

10(A) For *execlp*() and *execvp*():
PATH prefixes are separated by a colon ' :'. (See GA28 in 2.6.)

For *execl*(), *execv*(), *execle*(), and *execve*():
UNUSED.

11(A) For *execlp*() and *execvp*():
When a nonzero-length prefix is applied to a filename, then a '/' is inserted between the prefix and the filename. (See GA29 in 2.6.)

For *execl*(), *execv*(), *execle*(), and *execve*():
UNUSED.

12(A) For *execlp*() and *execvp*():
When a **PATH** prefix of zero length is specified by '::', then this indicates the current working directory. (See GA30 in 2.6.)

For *execl*(), *execv*(), *execle*(), and *execve*():
UNUSED.

13(A) For *execlp*() and *execvp*():
When **PATH** begins with an initial ':', then this indicates the current working directory. (See GA31 in 2.6.)

For *execl*(), *execv*(), *execle*(), and *execve*():
UNUSED.

14(A) For *execlp*() and *execvp*():
When PATH has a trailing ':', then this indicates the current working directory. (See GA32 in 2.6.)

For *execl*(), *execv*(), *execle*(), and *execve*():
UNUSED.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15(A) For *execlp*() and *execvp*():

The **PATH** environment is searched from beginning to end, and each colon-separated element is used as a prefix to a filename until an executable program by the specified name is found. (See GA33 in 2.6.)

For *execl*(), *execv*(), *execle*(), and *execve*():

UNUSED.

16(A) Upper- and lowercase letters in environment variable names and corresponding assigned values retain their unique identities. (See GA34 in 2.6.)

17(A) Environment variable names support the portable filename character set. (See GA35 in 2.6.)

D01(A) The results of an *execlp*() or *execvp*() call in the search of directories in the absence of environment variable **PATH** is stated in 3.1.2.2 of the PCD.1.

18(A) If {ARG_MAX} ≤ {PCTS_ARG_MAX}:

The exec elements accept arguments including environment data of {ARG_MAX} bytes allowing for NULL terminators, pointers, or alignment bytes as defined in the implementation.

Otherwise:

The exec elements accept arguments including environment data of {PCTS_ARG_MAX} bytes allowing for NULL terminators, pointers, or alignment bytes as defined in the implementation.

19(A) If {ARG_MAX} ≤ {PCTS_ARG_MAX}:

When the number of bytes in the argument and environment lists are less than or equal to {ARG_MAX}, then the arguments are available in the corresponding *main*() arguments, and the environment is available from the external variable *environ*.

Otherwise:

When the number of bytes in the argument and environment lists are less than or equal to {PCTS_ARG_MAX), then the arguments are available in the corresponding *main*() arguments, and the environment is available from the external variable *environ*.

20(A) For *execl*(), *execle*(), and *execlp*():

The argument list referenced by *argv* in *main*() of the new process image is obtained from the arguments *arg0, arg1,..., argn* in the calling process.

For *execv*(), *execve*(), and *execvp*():

The argument list referenced by *argv* in *main*() of the new process image is obtained from the argument *argv* in the calling process.

21(A) When the first filename component of the argument *path* or *file* is ".", and the pathname does not begin with a slash, then the exec type function resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

22(B) When the argument *path* or *file* points to the string "/", then the exec type function resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

See Reason 2 in Section 5. of POSIX. 3 {4}.

23(B) When the argument *path* or *file* points to the string "///", then the exec type function resolves the pathname to the root directory of the process.(See GA16 in 2.3.6.)

See Reason 2 in Section 5. of POSIX.3 {4}.

24(A) When the *path* or *file* argument points to a string beginning with a single slash or beginning with three or more slashes, then the exec type function resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

25(A) When the first filename component of the *path* or *file* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then the exec type function

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

26(B) When the argument *path* or *file* points to the string "F1/" and `F1` is a directory, then the `exec` type function resolves the pathname to F1, which is in the current working directory. (See GA19 in 2.3.6.)

See Reason 2 in Section 5. of POSIX.3 {4}.

27(B) When the argument *path* or *file* points to the string "F1//" and `F1` is a directory, then the `exec` type function resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

See Reason 2 in Section 5. of POSIX. 3 {4}.

28(A) When the argument *path* or *file* points to the string "F1/F2", then the `exec` type function resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

29(A) When the argument *path* or *file* points to the string "F1/./F2", then the `exec` type function resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

30(A) When the argument *path* or *file* points to the string "F1/../F1/F2", then the `exec` type function resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

31(A) When the argument *path* or *file* points to the string "F1//F2", then the `exec` type function resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

32(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

When the *path* or *file* argument pathname component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_- NO_TRUNC} is not supported, then the `exec` type function resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

33(A) On a call to `exec` elements, the pathname supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

34(A) On a call to `exec` elements, the pathname retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

35(A) For *execl*(), *execv*(), *execle*(), and *execve*():

When the function is granted execute access to *path* and is granted search access to the path prefix of *path*, then an [EACCES] error does not occur.

For *execlp*() and *execvp*():

When, after the appropriate element of **PATH** has been prefixed to *file*, the function is granted execute access to the resulting pathname, and when the function is granted search access to the path prefix of this pathname, then an [EACCES] error does not occur.

36(A) When file descriptors are opened in the calling process image, and they have the FD_CLOEXEC flag clear, then they remain open in the new process image. When they have the FD_CLOEXEC flag set, then they are closed.

37(A) When file descriptors remain open in the calling process image, then the file offset, file status, and file access mode of the open file description, as well as the file locks, remain unchanged by this call.

38(C) If {PCD_DIR_TYPE} is **TRUE** and the implementation supports the detection of [EBADF] for *closedir*() or *readdir*():

When a directory stream is open in the calling process image, then it is closed in the new process image.

39(A) When signals are set to signal-specific default)SIG_DFL) action in the calling process image, then they are set to SIG_DFL in the new process image.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

40(A) When signals are set to ignore signal (SIG_IGN) in the calling process image, then they are set to SIG_IGN in the new process image.

41(A) When signals are set to be caught in the calling process image, then they are set to the default action in the new process image.

42(A) When the set-user ID mode of the file being executed is not set, then the effective group ID of the new process remains the same as in the calling process.

43(A) When the set-group ID mode of the file being executed is not set, then the effective group ID of the new process image remains the same as in the calling process.

44(PCTS_CHMOD_SET_IDS?A:UNTESTED)

When the set-user ID mode of the file being executed is set, then the effective user ID of the new process is set to the user ID of the file executed, and the real user ID remains unchanged.

45(PCTS_CHMOD_SET_IDS?A:UNTESTED)

When the set-group ID mode of the file being executed is set, then the effective group ID of the new process is set to the group ID of the file executed, and the real group ID and supplementary group IDs remain unchanged.

46(PCTS_CHMOD_SET_IDS?C:UNTESTED)

If the behavior associated with {_POSIX_SAVED_IDS} is supported:
The effective user ID and effective group ID of the new process image are saved for use by *setuid* ().

47(A) The new process image of the calling process inherits the following attributes from the calling process:

- Process ID
- Parent process ID
- Process group ID
- Session membership
- Real user ID
- Real group ID
- Supplementary groups ID
- Time left until alarm clock signal
- Current working directory
- Root directory
- File mode creation mask
- Process signal mask
- Pending signals
- *tms_utime*
- *tms_stime*
- *tms_cutime*
- *tms_cstime*

D02(C) If the implementation documents whether there is support for inheritance of process characteristics for `exec` type functions not defined in POSIX.1 {3}:
The details are contained in 3.1.2.2 of the PCD.1.

48(A) When a call to `exec` is successful, then the *st_atime* field of the new process image file is marked for update.

D03(C) If the implementation documents whether the *st_atime* field for a process image file is marked for update when the system is able to find the file but the `exec` type function fails:
The details are contained in 3.1.2.2 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.1.2.3 Returns

R01 When a call to an exec element completes unsuccessfully, then a value of (*int*)–1 is returned and *errno* is set to indicate the error. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed, signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified. (See Assertions 49-60 in 3.1.2.4.)

3.1.2.4 Errors

49({ARG_MAX}≤{PCTS_ARG_MAX}?A:UNTESTED)

When the number of bytes used in the argument list and the environment, including NULL terminators, pointers, and alignment bytes, is greater than the system-imposed limit for exec ({ARG_MAX} bytes), then a call to these functions returns a value of (*int*)–1 and sets *errno* to [E2BIG]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

50(A) When search permission is denied for a directory listed in the *path* prefix of the new process image file, then a call to exec returns a value of (*int*)–1 and sets *errno* to [EACCES]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

51(A) When the new process image file denies execution permission, then a call to exec returns a value of (*int*)–1 and sets *errno* to [EACCES]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

52(D) If there exists a type of file for which the implementation does not support execution of files of this type:
A call to exec with a file of that type returns a value of (*int*)-1 and sets *errno* to [EACCES]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

See Reason 2 in Section 5. of POSIX.3 [4].

D04(C) If the implementation allows the execution of irregular files, and this is documented:
The details are contained in 3.1.2.4 of the PCD.1.

53(A) For *execl*(), *execv*(), *execle*(), and *execve*():
If{PATH_MAX}≤{PCTS_PATH_MAX}:
When the length of the *path* argument exceeds the maximum number of characters in a pathname, then a call to exec returns a value of (*int*)–1 and sets *errno* to [ENAMETOOLONG]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not dosed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Otherwise:

When the length of the path argument is {PCTS_PATH_MAX}, then a call to exec is successful.

For *execlp*() and *execvp*():

If {PATH_MAX}≤{PCTS_PATH_MAX}:

When the length of each of the pathnames generated by prefixing an element of the environment variable **PATH** to the file argument exceeds the maximum number of characters in a pathname, then a call to exec returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

Otherwise:

When the length of the file argument is {PCTS_PATH_MAX}, then a call to exec is successful.

54({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported:

For *execl*(), *execv*(), *execle*(), and *execve*():

When the length of a pathname component is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to exec returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

For *execlp*() and *execvp*():

When each of the pathnames generated by prefixing an element of the environment variable **PATH** to the file argument contains a pathname component that is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to exec returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

55({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported:

For *execl*(), *execv*(), *execle*(), and *execve*():

When the length of a pathname component equals {PCTS_NAME_MAX}, then a call to an exec type function succeeds.

For *execlp*() and *execvp*():

When the length of all the pathnames generated by prefixing an element of the environment variable **PATH** to the file argument contain a pathname component of size equal to {PCTS_NAME_MAX}, then a call to an exec type function succeeds.

56(A) For *execl*(), *execv*(), *execle*(), and *execve*():

When a component of the *path* argument does not exist, then a call to exec returns a value of (*int*)-1 and sets *errno* to [ENOENT]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variabl **environ**, the pointers it contains, or the strings to which they point are not modified.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

For *execlp*() and *execvp*():

When none of the pathnames generated by prefixing an element of the environment variable **PATH** to the file argument exists, then a call to exec returns a value of (*int*)-1 and sets *errno* to [ENOENT]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

57(A) When the *path* or *file* argument points to an empty string, then a call to exec returns a value of (*int*)-1 and sets *errno* to [ENOENT]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

58(A) For *execl*(), *execv*(), *execle*(), and *execve*():

When a component of the *path* prefix of the new process image file is not a directory, then a call to exec returns a value of (*int*)-1 and sets *errno* to [ENOTDIR]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

For *execlp*() and *execvp*():

When a component of the path prefix of each of the pathnames generated by prefixing an element of the environment variable **PATH** to the *file* argument is not a directory, then a call to exec returns a value of (*int*)-1 and sets *errno* to [ENOTDIR]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

59(A) For *execl*(), *execv*(), *execle*(), and *execve*():

When the new process image file has the required permission but is not in the proper format, then a call to exec returns a value of (*int*)-1 and sets *errno* to [ENOEXEC]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

For *execlp*() and *execvp*():

UNUSED.

D05(C) If the implementation supports the detection of [ENOMEM] for exec:

The conditions under which [ENOMEM] occurs for exec type functions is contained in 3.1.2.4 of the PCD.1. (See DGA02 in 2.4.)

60(B) If the implementation supports the detection of [ENOMEM] for exec:

When the new process image requires more memory than is allowed by the hardware or by system-imposed memory management constraints, then a call to exec returns a value of (*int*)-1 and sets *errno* to [ENOMEM]. Neither the *argv*[] and *envp*[] arrays of pointers nor the strings to which those array elements point are modified. File descriptors marked close-on-exec are not closed; signals set to be caught are not set to default; neither the effective user ID nor the effective group ID of the current process is changed; and the value of the global variable **environ**, the pointers it contains, or the strings to which they point are not modified.

See Reason 1 in Section 5. of POSIX.3 [4].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Otherwise:

When the new process image requires more memory than is allowed by the hardware or by system-imposed memory management constraints, then a call to exec is successful (unless a different error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 [4].

3.2 Process Termination

GA40 For *_exit*() and process termination by a signal:

The process ceases the execution of subsequent program statements.

For *exit*():

If the function *atexit*() is supported:

When the process returns from the functions previously registered by *atexit*(), then the process ceases the execution of subsequent program statements.

Otherwise:

The process ceases the execution of subsequent program statements.

For *abort*():

When the SIGABRT signal is not being caught or the signal handler associated with SIGABRT returns, then the process ceases the execution of subsequent program statements.

GA41 For *_exit*(), *exit*(), *abort*(), and process termination by a signal:

All open file descriptors are closed.

GA42 For *_exit*(), *exit*(), *abort*(), and process termination by a signal:

All open directory streams are closed.

GA43 For *_exit*() and process termination by a signal:

When the parent process of the calling process is waiting for this process, then it is notified of the calling process termination and the low-order 8 b of status are made available to it.

For *exit*():

If the function *atexit*() is supported:

When the process returns from the functions previously registered by *atexit*(), and the parent process of the calling process is waiting for this process, then it is notified of the calling process termination and the low-order 8 b of status are made available to it.

Otherwise:

When the process calls *exit*(), and the parent process of the calling process is waiting for this process, then it is notified of the calling process termination and the low-order 8 b of status are made available to it.

For *abort*():

When the SIGABRT signal is not being caught or the signal handler associated with SIGABRT returns, and when the parent process of the calling process is waiting for this process, then it is notified of the calling process termination and the low-order 8 b of status are made available to it.

GA44 For *_exit*() and process termination by a signal:

When the parent process of the calling process is not waiting for the process, then the exit status code is saved for return to the parent process whenever the parent process executes a subsequent *wait*() or *waitpid*() for this process.

For *exit*():

If the function *atexit*() is supported:

When the process returns from the functions previously registered by *atexit*(), and the parent process of the calling process is not waiting for the process, then the exit status code is saved for

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

return to the parent process whenever the parent process executes a subsequent *wait*() or *waitpid*() for this process.

Otherwise:

When the process calls *exit*(), and the parent process of the calling process is not waiting for the process, then the exit status code is saved for return to the parent process whenever the parent process executes a subsequent *wait*() or *waitpid*() for this process.

For *abort*():

When the SIGABRT signal is not being caught or the signal handler associated with SIGABRT returns, and when the parent process of the calling process is not waiting for the process, then the exit status code is saved for return to the parent process whenever the parent process executes a subsequent *wait*() or *waitpid*() for this process.

GA45 For *_exit*() and process termination by a signal:

When a parent process that is not a controlling process terminates without waiting for its child processes to terminate, then the remaining child processes are not terminated and are assigned a new parent process ID.

For *exit*():

If the function *atexit*() is supported:

When the process returns from the functions previously registered by *atexit*(), and a parent process that is not a controlling process terminates without waiting for its child processes to terminate, then the remaining child processes are not terminated and are assigned a new parent process ID.

Otherwise:

When the process calls *exit*(), and a parent process that is not a controlling process terminates without waiting for its child processes to terminate, then the remaining child processes are not terminated and are assigned a new parent process ID.

For *abort*():

When the SIGABRT signal is not being caught or the signal handler associated with SIGABRT returns, and when a parent process that is not a controlling process terminates without waiting for its child processes to terminate, then the remaining child processes are not terminated and are assigned a new parent process ID.

GA46 For *_exit*() and process termination by a signal:

If either the behavior associated with {_POSIX_JOB_CONTROL} is supported or SIGCHLD is supported:

When a process terminates, then a SIGCHLD signal is sent to its parent process.

For *exit*():

If the function *atexit*() is supported and either the behavior associated with {_POSIX_JOB_CONTROL} is supported or SIGCHLD is supported:

When the process returns from the functions previously registered by *atexit*(), then a SIGCHLD signal is sent to its parent process.

If the function *atexit*() is not supported and either the behavior associated with {_POSIX_JOB_CONTROL} is supported or SIGCHLD is supported:

When the process calls *exit*(), then a SIGCHLD signal is sent to its parent process.

For *abort*():

If either the behavior associated with {_POSIX_JOB_CONTROL} is supported or SIGCHLD is supported:

When the SIGABRT signal is not being caught or the signal handler associated with SIGABRT returns, then a SIGCHLD signal is sent to its parent process.

GA47 For *_exit*() and process termination by a signal:

When a controlling process terminates, then a SIGHUP signal is sent to each process in the foreground process group of the controlling terminal belonging to the controlling process. The

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

remaining child processes, which are not terminated upon delivery of the SIGHUP signal, are assigned a new parent process ID.

For *exit*():

If the function *atexit*() is supported:

When a controlling process returns from the functions previously registered by *atexit*(), then a SIGHUP signal is sent to each process in the foreground process group of the controlling terminal belonging to the controlling process. The remaining child processes, which are not terminated upon delivery of the SIGHUP signal, are assigned a new parent process ID.

Otherwise:

When a controlling process calls *exit*(), then a SIGHUP signal is sent to each process in the foreground process group of the controlling terminal belonging to the controlling process. The remaining child processes, which are not terminated upon delivery of the SIGHUP signal, are assigned a new parent process ID.

For *abort*():

When a controlling process terminates and the SIGABRT signal is not being caught or the signal handler associated with SIGABRT returns, then a SIGHUP signal is sent to each process in the foreground process group of the controlling terminal belonging to the controlling process. The remaining child processes, which are not terminated upon delivery of the SIGHUP signal, are assigned a new parent process ID.

GA48 For *_exit*() and process termination by a signal:

When the calling process, which is a controlling process, terminates, then the controlling terminal associated with the session is disassociated from the session.

For *exit*():

If the function *atexit*() is supported:

When the calling process, which is a controlling process, returns from the functions previously registered by *atexit*(), then the controlling terminal associated with the session is disassociated from the session.

Otherwise:

When the calling process calls *exit*() from a controlling process, then the controlling terminal associated with the session is disassociated from the session.

For *abort*():

When the SIGABRT signal is not being caught or the signal handler associated with SIGABRT returns and the calling process, which is a controlling process, terminates, then the controlling terminal associated with the session is disassociated from the session.

GA49 For *_exit*() and process termination by a signal:

If the behavior associated with {POSIX_JOB_CONTROL} is supported:

When termination of the process causes a process group to be orphaned, and any member of the newly orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal is sent to each process in the newly orphaned process group.

For *exit*():

If the function *atexit*() is supported and the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process returning from the functions previously registered by *atexit*() causes a process group to be orphaned, and any member of the newly orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal is sent to each process in the newly orphaned process group.

If the function *atexit*() is not supported and the behavior associated with {_POSIX_JOB_CONTROL} is supported:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the process calling *exit*() causes a process group to be orphaned, and any member of the newly orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal is sent to each process in the newly orphaned process group.

For *abort*():

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the SIGABRT signal is not being caught or the signal handler associated with SIGABRT returns, and when termination of the process causes a process group to be orphaned, and when any member of the newly orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal is sent to each process in the newly orphaned process group.

GA50 For *abort*():

When the SIGABRT signal is not being caught or the signal handler associated with SIGABRT returns, then on a call to *abort*() the status made available to *wait*() or *waitpid*() is that of a process terminated by the SIGABRT signal.

There are no assertions specific to this subclause.

3.2.1 Wait for Process Termination

3.2.1.1 *wait*() (3.2.1)

3.2.1.1.1 Synopsis (3.2.1.1)

01(A) If the implementation provides C Standard {2} support:

When the header <sys/wait.h> is included, then the function prototype `pid_t wait(int *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <sys/wait.h> is included, then the function *wait*() is declared with the result type *pid_t*. (See GA36 in 2.7.3.)

02(C) If *wait*() is defined as a macro when the header <sys/wait.h> is included:

When the macro *wait*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *pid_t*. (See GA37 in 2.7.3.)

03(C) If *wait*() is defined as a macro in the header <sys/wait.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.2.1.1.2 Description (3.2.1.2)

D01(C) If the order in which status is reported when status information is available for two or more child processes is documented:

The details are contained in 3.2.1.2 of the PCD.1.

04(A) A call to *wait*() suspends execution of the calling process until status information for one of its terminated child processes is available or until delivery of a signal whose action is either to execute a signal-catching function or to terminate the calling process.

05(A) When status information is available prior to a call to *wait*(), then return is immediate.

06(A) When the status of a child is or becomes available, then the value returned by *wait*() is the value of the process ID of the child process.

Testing Requirements:

Test with *stat_loc* set to NULL and *stat_loc* set to a valid address.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

R01 When the argument *stat_loc* is not NULL then on a successful return from *wait*() the status information is stored in the location pointed to by *stat_loc*. (See Assertions 9, 11, and 12 in 3.2.1.1.2.)

07(A) When the status returned is from a terminated child process that returned a value of zero from *main*(), or when a value of zero is passed as the status argument to *_exit*() or *exit*(), then the value stored at the location pointed to by *stat_loc* is zero.

08(A) When the header <sys/wait.h> is included, the macros WIFEXITED, WEXITSTATUS, WIFSIGNALED, WTERMSIG, WIFSTOPPED, and WSTOPSIG are defined.

09(A) When the header <sys/wait.h> is included and the stratus is returned for a child process that either returned from *main*(), called *_exit*(), or called *exit*(), then the macro WIFEXITED(**stat_loc*) evaluates to a nonzero value, and the macros WIFSIGNALED(**stat_loc*) and WIFSTOPPED(**stat_loc*) evaluate to zero.

10(A) When the header <sys/wait.h> is included and the value of WIFEXITED(**stat_loc*) is nonzero—where *stat_loc* is the status argument populated by an invocation of *wait*()—then the macro WEXITSTATUS(**stat_loc*) evaluates to the low-order 8 b of the status argument that the child process passed to *_exit*() or *exit*(), or to the low-order 8 b of the value the child process returned from *main*().

11(A) When the header <sys/wait.h> is included and the status was returned for a child process that terminated due to the receipt of a signal that was not caught, then the macro WIFSIGNALED(**stat_loc*) evaluates to a nonzero value, and the macros WIFEXITED(**stat_loc*) and WIFSTOPPED(**stat_loc*) evaluate to zero.

12(A) When the header <sys/wait.h> is included and the value of WIFSIGNALED(**stat_loc*) is nonzero—where *stat_loc* is the status argument populated by an invocation of *wait*()—then the macro WTERMSIG(**stat_loc*) evaluates to the number of the signal that caused the termination of the child process.

13(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

A call to *wait*() does not return when one of the child processes of the calling process is stopped.

D02(C) If an implementation supports additional circumstances under which *wait*() or *waitpid*() report status, and this is documented:

The details on the circumstances and the meaning of the reported status bytes are contained in 3.2.1.2 of the PCD.1.

3.2.1.1.3 Returns (3.2.1.3)

R02 When a call to *wait*() completes successfully, then the value returned is equal to the process ID of the child process whose status is reported. (See Assertion 6 in 3.2.1.1.2.)

R03 When the delivery of a signal causes *wait*() to complete unsuccessfully, *wait*() returns a value of (*pid_t*)-1 and sets *errno* to [EINTR]. (See Assertion 15 in 3.2.1.1.4.)

R04 When a call to *wait*() completes unsuccessfully, then a value of (*pid_t*)−1 is returned and sets *errno* to indicate the error. (See Assertions 14 and 15 in 3.2.1.1.4.)

3.2.1.1.4 Errors (3.2.1.4)

14(A) When the calling process has no existing unwaited-for child processes, then a call to *wait*() returns a value of (*pid_t*)-1 and sets *errno* to [ECHILD].

15(A) When a call to *wait*() is interrupted by a signal, then *wait*() returns a value of (*pid_t*)-1 and sets *errno* to [EINTR]. The wait statuses of all child processes of the calling process remain available for subsequent calls to *wait*() or *waitpid*().

D03(C) If the value of the location pointed to by *stat_loc* when a call to *wait*() or *waitpid*() returns an *errno* of [EINTR] is documented:

The details are contained in 3.2.1.4 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.2.1.2 *waitpid*() (3.2.1)

3.2.1.2.1 Synopsis (3.2.1.1)

01(A) If the implementation provides C Standard {2} support:

When the header <sys/wait.h> is included, then the function prototype `pid_t waitpid(pid_t, int *, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <sys/wait.h> is included, then the function *waitpid*() is declared with the result type *pid_t*. (See GA36 in 2.7.3.)

02(C) If *waitpid*() is defined as a macro when the header <sys/wait .h> is included:

When the macro *waitpid*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *pid_t*. (See GA37 in 2.7.3.)

03(C) If *waitpid*() is defined as a macro in the header <sys/wait.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.2.1.2.2 Description (3.2.1.2)

04(A) A call to *waitpid(-1, stat_loc, 0)* suspends execution of the calling process until status information for any one of its terminated child processes is available or until delivery of a signal whose action is either to execute a signal-catching function or to terminate the calling process.

05(A) When status information for any child of the calling process is available prior to a call to *waitpid(-1, stat_loc, options)*, then return is immediate.

06(A) When the status of any child of the calling process is or becomes available, then the value returned by *waitpid(-1, stat_loc, options)* is the value of the process ID of the child process.

Testing Requirements:

Test with *stat_loc* set to NULL and *stat_loc* set to a valid address.

07(A) When the argument *pid* is greater than zero and is equal to the process ID of a child of the calling process, then a call to *waitpid(pid, stat_loc, options)* returns when the process specified by *pid* has terminated.

08(A) When the argument *pid* is equal to zero, then a call to *waitpid(pid, stat_loc, options)* returns when any child process that is a member of the same process group as the calling process has terminated.

09(A) When the argument *pid* is less than −1, then a call to *waitpid(pid, stat_loc, options)* returns when any child process with a process group ID equal to the absolute value of the argument *pid* has terminated.

10(A) When the header <sys/wait.h> is included, then the flags WNOHANG and WUNTRACED are defined as bitwise exclusive integer values.

11(A) When status is not immediately available for one of the child processes specified by *pid*, and the options argument specifies the WNOHANG flag, then a call to *waitpid(pid, stat_loc, options)* does not suspend execution of the calling process, and a value of zero is returned.

Testing Requirements:

If {_POSIX_JOB_CONTROL} is supported, test with options set to WNOHANG | WUNTRACED in addition to testing with options set to WNOHANG.

12(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

A call to *waitpid(pid, stat_loc, WUNTRACED)* or to *waitpid(pid, stat_loc, WNOHANG | WUNTRACED)* reports to the calling process the status of any child process specified by *pid* that is stopped and has not been reported since it stopped, or the status of any child process that has exited.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

R01 When the argument *stat_loc* is not NULL, then on a successful return from a call to *waitpid(pid, stat_loc, options)*, the status information is stored in the location pointed to by *stat_loc*. (See Assertions 14, 16, and 17 in 3.2.1.2.2.)

13(A) When the status returned to *waitpid(pid, stat_loc, options)* is from a terminated child process that returned a value of zero from *main*() or passed a value of zero as the status argument to *_exit*() or *exit*(), then the value stored at the location pointed to by *stat_loc* is zero.

14(A) When the header <sys/wait.h> is included and the status is returned for a child process by returning from *main*() or by a call to *_exit*() or *exit*(), then the macro WIFEXITED(**stat_loc*) evaluates to a nonzero value, and the macros WIFSIGNALED(**stat_loc*) and WIFSTOPPED(**stat_loc*) evaluate to zero.

15(A) When the header <sys/wait.h> is included and the value of WIFEXITED(**stat_loc*) is nonzero, then the macro WEXITSTATUS(**stat_loc*) evaluates to the low-order 8 b of the status argument that the child process passed to *exit*() or *exit*(), or to the low-order 8 b of the value the child process returned from *main*().

16(A) When the header <sys/wait.h> is included and the status was returned for a child process that terminated due to the receipt of a signal that was not caught, then the macro WIFSIGNALED(**stat_loc*) evaluates to a nonzero value, and the macros WIFEXITED(**stat_loc*) and WIFSTOPPED(**stat_loc*) evaluate to zero.

17(A) When the header <sys/wait.h> is included and the value of WIFSIGNALED(**stat_loc*) is nonzero, then the macro WTERMSIG(**stat_loc*) evaluates to the number of the signal that caused the termination of the child process.

18(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the header <sys/wait.h> is included and the status was returned for a child process that is currently stopped, then the macro WIFSTOPPED(**stat_loc*) evaluates to a nonzero value, and the macros WIFEXITED(**stat_loc*) and WIFSIGNALED(**stat_loc*) evaluate to zero.

19(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the WUNTRACED flag is clear, then a call to *waitpid*() does not return the status of its stopped child processes.

20(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the header <sys/wait.h> is included and the value of WIFSTOPPED(**stat_loc*) is nonzero, then the macro WSTOPSIG(**stat_loc*) evaluates to the number of the signal that caused the process to stop.

3.2.1.2.3 Returns (3.2.1.3)

R02 When a call to *waitpid*() completes successfully, then the value returned is equal to the process ID of the child process whose status is reported. (See Assertion 6 in 3.2.1.2.2.)

R03 When the delivery of a signal causes *waitpid*() to complete unsuccessfully, *waitpid*() returns a value of (*pid_t*)−1 and sets *errno* to [EINTR]. (See Assertion 26 in 3.2.1.2.4.)

R04 When status is not immediately available for one of the child processes specified by pid and the options argument specifies the WNOHANG flag, then a call to *waitpid(pid, stat_loc, options)* does not suspend execution of the calling process and a value of zero is returned. (See Assertion 11 in 3.2.1.2.4.)

R05 When a call to *waitpid*() completes unsuccessfully, then a value of (*pid_t*)−1 is returned and sets *errno* to indicate the error. (See Assertions 21-27 in 3.2.1.2.4.)

3.2.1.2.4 Errors (3.2.1.4)

21(A) When the process or process group specified by *pid* does not exist, then a call to *waitpid(pid, stat_loc, options)* returns a value of (*pid_t*)−1 and sets *errno* to [ECHILD]. The wait statuses of all child processes of the calling process remain available for subsequent calls to *wait*() or *waitpid*().

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

22(A) When *pid* is equal to −1 and the calling process does not have any child processes, then a call to *waitpid(pid, stat_loc, options)* returns a value of (*pid_t*)−1 and sets *errno* to [ECHILD].

23(A) When pid is equal to 0 and the calling process does not hav any child processes in its process group, then a call to *waitpid(pid, stat_loc, options)* returns a value of (*pid_t*)−1 and sets *errno* to [ECHILD]. The wait statuses of all child processes of the calling process remain available for subsequent calls to *wait*() or *waitpid*().

24(A) When *pid* is less than −1 and the calling process does not have any child processes in the process group specified by pid, then a call to *waitpid(pid, stat_loc, options)* returns a value of (*pid_t*)−1 and sets *errno* to [ECHILD].

25(A) When *pid* is greater than zero and the process *pid* is not a child of the calling process, then a call to *waitpid(pid, stat_loc, options)* returns a value of (*pid_t*)−1 and sets *errno* to [ECHILD]. The wait status of the process *pid* remains available for subsequent calls to *wait*() or *waitpid*() by its parent process.

26(A) When a call to *waitpid(pid, stat_loc, options)* is interrupted by a signal, then *waitpid*() returns a value of (*pid_t*)−1 and sets *errno* to [EINTR]. The wait statuses of all child processes of the calling process remain available for subsequent calls to *wait*() or *waitpid*().

27(A) When the value of the options argument is not valid then a call to *waitpid(pid, stat_loc, options)* returns a value of (*pid_t*)−1 and sets *errno* to [EINVAL]. The wait statuses of all child processes of the calling process remain available for subsequent calls to *wait*() or *waitpid*().

3.2.2 *_exit*()

3.2.2.1 Synopsis

01(C) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `void_exit(int)` is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *_exit*() is contained in 3.2.2.1 of the PCD.1.

02(D) If the implementation provides C Standard {2} support and *_exit*() is defined as a macro when the header <unistd.h> is included:
When the macro *_exit*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type void. (See GA37 in 2.7.3.)

See Reason 2 in Section 5. of POSIX.3 {4}.

03(C) If *_exit*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 in 1.3.3.)

3.2.2.2 Description

04(A) On a call to *_exit*(), the process ceases the execution of subsequent program statements. (See GA40 in 3.2.)

05(A) On a call to *_exit*(), all open file descriptors are closed. (See GA41 in 3.2.)

06(B) On a call to *_exit*(), all open directory streams are closed. (See GA42 in 3.2.)

See Reason 2 in Section 5. of POSIX.3 {4}.

07(A) When the parent process of the calling process is executing a *wait*() or *waitpid*(), then on a call to *_exit*() by the child, the parent is notified of the calling process termination, and the low-order 8 b of status are made available to the parent. (See GA43 in 3.2.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

08(A) When the parent process of the calling process is not waiting for the process, then on a call to *_exit*() the exit status code is saved for return to the parent process whenever the parent process executes a subsequent *wait*() or *waitpid*() for this process. (See GA44 in 3.2.)

09(A) When a parent process that is not a controlling process terminates by calling *_exit*() without waiting for its child processes to terminate, then the remaining child processes are not terminated and are assigned a new parent process ID. (See GA45 in 3.2.)

D02(A) The new parent process ID assigned to children of a terminated process is stated in 3.2.2 of the PCD.1.

10(C) If either the behavior associated with {_POSIX_JOB_CONTROL} is supported or SIGCHLD is supported:
When a process terminates by calling *_exit*() a SIGCHLD signal is sent to its parent process. (See GA46 in 3.2.)

11(PCTS_GTI_DEVICE?A:UNTESTED)

When a process that is a controlling process terminates by calling *_exit*(), then a SIGHUP signal is sent to each process in the foreground process group of the controlling terminal belonging to the controlling process. The remaining child processes, which are not terminated upon delivery of the SIGHUP signal, are assigned a new parent process ID. (See GA47 in 3.2.)

12(PCTS_GTI__DEVICE?A:UNTESTED)

When the calling process is a controlling process, process termination causes the controlling terminal associated with the session to be disassociated from the session. (See GA48 in 3.2.)

13(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When termination of the process causes a process group to be orphaned, and any member of the newly orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal is sent to each process in the newly orphaned process group. (See GA49 in 3.2.)

3.2.2.3 Returns

R01 A call to *_exit*() does not return. (See Assertions 4–13 in 3.2.2.2.)

3.3 Signals

3.3.1 Signal Concepts

3.3.1.1 Signal Names

01(A) When the header `<signal.h>` is included, then the *sigset_t* type and the structure *sigaction* are declared, and the symbolic constants SIG_DFL and SIG_IGN are defined.

02(A) The constants SIG_DFL and SIG_IGN are of a type compatible with *void(*)()*.

03(B) The values of SIG_DFL and SIG_IGN match no declarable function.

See Reason 3 in Section 5. of POSIX.3 {4}.

04(A) When the header `<signal.h>` is included, then the symbolic signal constants SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU are defined.

05(A) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
The signals SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU have distinct positive integral values.

Otherwise:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The signals SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM SIGUSR1, and SIGUSR2 have distinct positive integral values.

06(A) The default action for the signals SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, and SIGUSR2 is abnormal termination of the process.

07(A) When a signal is received whose action is to terminate the process, then the process is terminated. (See GA40 in 3.2.)

08(A) When a signal is received whose action is to terminate the process, then all open file descriptors in the process are closed. (See GA41 in 3.2.)

09(B) When a signal is received whose action is to terminate the process, then all open directory streams in the process are closed. (See GA42 in 3.2.)

See Reason 2 in Section 5. of POSIX.3 [4].

10(A) When the parent process of the process terminated by a signal is waiting for the process, then it is notified of the calling process termination, and the signal number that caused the termination is made available to it. (See GA43 in 3.2.)

11(A) When the parent process of the process terminated by a signal is not waiting for the process, then the signal number that caused the termination is saved for return to the parent process whenever the parent process executes a subsequent *wait*() or *waitpid*() for this process. (See GA44 in 3.2.)

12(A) When a parent process that is not a controlling process terminates without waiting for its child processes to terminate, then the remaining child processes are not terminated and are assigned a new parent process ID. (See GA45 in 3.2.)

13(C) If either the behavior associated with {_POSIX_JOB_CONTROL} is supported or SIGCHLD is supported:
When a process terminates, a SIGCHLD signal is sent to its parent process. (See GA46 in 3.2.)

14(PCTS_GTI_DEVICE?A:UNTESTED)

When a process that is a controlling process terminates, then a SIGHUP signal is sent to each process in the foreground process group of the controlling terminal belonging to the controlling process. The remaining child processes, which are not terminated upon delivery of the SIGHUP signal, are assigned a new parent process ID. (See GA47 in 3.2.)

15(PCTS_GTI_DEVICE?A:UNTESTED)

When the calling process is a controlling process, process termination causes the controlling terminal associated with the session to be disassociated from the session. (See GA48 in 3.2.)

16(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When termination of the process causes a process group to be orphaned, and any member of the newly orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal is sent to each process in the newly orphaned process group. (See GA49 in 3.2.)

17(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
The default action for SIGCHLD is to ignore the signal.

18(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
The default action for SIGCONT is to continue the process if it is stopped; otherwise, the default action is to ignore the signal.

19({POSIX_JOB_CONTROL}?C:UNTESTED)

If SIGSTOP is supported:
The default action for SIGSTOP is to stop the process.

20({POSIX_JOB_CONTROL}?C:UNTESTED)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860200 917 ■

If SIGTSTP is supported:
The default action for SIGTSTP is to stop the process.

21({POSIX_JOB_CONTROL}?C:UNTESTED)

If SIGTTIN is supported:
The default action for SIGTTIN is to stop the process.

22({POSIX_JOB_CONTROL}?C:UNTESTED)

If SIGTTOU is supported:
The default action for SIGTTOU is to stop the process.

D01(C) If the behavior associated with {_POSIX_JOB_CONTROL} is not supported and the support of the signals defined in Table 3-2 of POSIX.1 {3} is documented:
The details are contained in 3.3.1.1 of the PCD.1.

D02(C) if the implementation supports additional signals, and this is documented:
The details are contained in 3.3.1.1 of the PCD.1.

3.3.1.2 Signal Generation and Delivery

23(A) When a signal, whose action is not set to ignore the signal, is blocked and the signal is generated, then the signal remains pending until it is either unblocked or ignored. (See Section B.3.3.1.2.)

24(A) When a signal, whose action is set to ignore the signal, is blocked and the signal is generated, then either it is discarded immediately or it remains pending until either it is unblocked or its action is again set to ignore the signal.

D03(C) If a signal is discarded immediately upon generation or remains pending when the action associated with a blocked signal is SIG_IGN, and this is documented:
The details are confined in 3.3.1.2 of the PCD.1.

25(A) The action taken on delivery of a signal is determined at the time of delivery, taking into account *sigaction*() calls while the signal is pending.

D04(A) Whether a signal is delivered more than once if a subsequent occurrence of a pending signal is generated is described in 3.3.1.2 of the PCD.1.

D05(C) If the order in which multiple simultaneously pending signals are delivered to a process is documented:
The details are contained in 3.3.1.2 of the PCD.1.

26(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process, then any pending SIGCONT signals for that process are discarded.

27(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When SIGCONT is generated for a process, then the pending stop signals (SIGTSTP, SIGTTIN, SIGTTOU) for that process are discarded.

28(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When SIGCONT is generated for a process that is stopped, then the process continues, even when the SIGCONT signal is blocked or ignored.

29(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When SIGCONT is generated for a process that is stopped, and the SIGCONT signal is both blocked and not ignored, then it remains pending until it is either unblocked or a stop signal is generated for the process.

D06(A) When signals are generated under conditions not defined by POSIX.1 {3}, the details are contained in 3.3.1.2 of the PCD.1 or in the subclause of the PCD.1 that corresponds to the subclause in POSIX.1 {3} where the function that provokes the signal is defined.

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.3.1.3 Signal Actions

30(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a process is stopped, then its execution is suspended.

31(A) When the *main*() routine of a new process image is entered, then all signals that are set to the default action or are to be caught in the calling image are set to SIG_DFL in the new process image, and all signals set to be ignored in the calling process image are set to SIG_IGN in the new process image.

32(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a process whose parent has not set the SA_NOCLDSTOP flag stops, a SIGCHLD signal shall be generated for the parent process.

33(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a process is stopped and a SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, or SIGCHLD signal is sent to the process, then the signal is not delivered until the process is continued.

34(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a process is stopped and a SIGKILL is sent to the process, then the signal terminates the receiving process.

35(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a process that is a member of an orphaned process group is sent a SIGTSTP, SIGTTIN, or SIGTTOU signal, then the process is not stopped by this signal, and the signal is discarded.

36(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the action for SIGCHLD is set to SIG_DFL while a SIGCHLD signal is pending, then the pending SIGCHLD signal is discarded.

R01 The system does not allow the action for the signal SIGKILL to be set to SIG_IGN. (See Assertion 20 in 3.3.4.4.)

R02 The system does not allow the action for the signal SIGSTOP to be set to SIG_IGN. (See Assertion 21 in 3.3.4.4.)

37(A) When a signal whose action is set to SIG_IGN is delivered, then the delivery of the signal does not affect the process.

D07(C) If the behavior of a process after it ignores a SIGFPE, SIGILL, or SIGSEGV signal that was not generated by the *kill*() or *raise*() functions is documented:
The details are contained in 3.3.1.3 of the PCD.1.

38(A) When the action for a pending signal that is also blocked is set to SIG_IGN, then the pending signal is discarded.

39(B) When the action for a pending signal that is not blocked is set to SIG_IGN, then the pending signal is discarded.

See Reason 1 in Section 5. of POSIX.3 [4].

D08(C) If the action taken when a process sets the action for the SIGCHLD signal to SIG_IGN is documented:
The details are contained in 3.3.1.3 of the PCD.1.

40(A) When any one of the signals SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, or SIGUSR2 is sent to a process whose action is set to catch the signal, then the receiving process executes the signal-catching function at the address specified, and the signal number is passed as the single argument to this function.

41(C) If the behavior associated with {_POSIX_JOB CONTROL} is supported:
When any one of the signals SIGCHLD, SIGCONT, SIGTSTP, SIGTTIN, or SIGTTOU is sent to a process whose action is set to catch the signal, then the receiving process executes the signal-

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

catching function at the address specified, and the signal number is passed as the signal argument to this function.

42(A) When a process returns from the execution of a signal-catching function, then the process resumes execution from the point at which it was interrupted.

D09(C) If the action taken when a process returns normally from a signal-catching function for a SIGFPE, SIGILL, or SIGSEGV signal that was not generated by the *kill*() or *raise*() function is documented:
The details are contained in 3.3.1.3 of the PCD.1.

43(A) The system does not allow a process to catch the signal SIGKILL.

44(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
The system does not allow a process to catch the signal SIGSTOP.

D10(C) If the action taken when a process establishes a signal-catching function for the SIGCHLD signal while it has a terminated child process for which it has not waited is documented:
Whether a SIGCHLD signal is generated is stated in 3.3.1.3 of the PCD.1.

45(A) The functions *fcntl*(), *kill*(), *open*(), *pause*(), *read*(), *sleep*(), *sigprocmask*(), *sigsuspend*() *wait*(), *waitpid*(), and *write*() are reentrant with respect to signals. That is, they can be called without restriction from a signal-catching function.

46(PCTS_GTI_DEVICE && PCTS_GTI_BUFFERS_OUTPUT?A:UNTESTED)

The function *tcdrain*() is reentrant with respect to signals. That is, *tcdrain*() can be called without restriction from a signal-catching function.

47(B) The functions *_exit*(), *access*(), *alarm*(), *cfgetispeed*(), *cfgetospeed*(), *cfsetispeed*(), *cfsetospeed*(), *chdir*(), *chmod*(), *chown*(), *close*(), *creat*(), *dup*(), *dup2*(), *execle*(), *execve*(), *fork*(), *fstat*(), *getegid*(), *geteuid*(), *getgid*(), *getgroups*(), *getpgrp*(), *getpid*(), *getppid*(), *getuid*(), *link*(), *lseek*(), *mkdir*(), *mkfifo*(), *pathconf*(), *pipe*(), *rename*(), *rmdir*(), *setgid*(), *setpgid*(), *setsid*(), *setuid*(), *sigaction*(), *sigaddset*(), *sigdelset*(), *sigemptyset*(), *sigfillset*(), *sigpending*(), *sigismember*(), *stat*(), *sysconf*(), *tcflow*(), *tcflush*(), *tcgetattr*(), *tcgetpgrp*(), *tcsendbreak*(), *tcsetattr*(), *tcsetpgrp*(), *time*(), *times*(), *umask*(), *uname*(), *unlink*(), and *utime*() are reentrant with respect to signals.

See Reason 3 in Section 5. of POSIX.3 {4}.

48(B) When a function defined by POSIX.1 {3} or by the C Standard {2} is interrupted by a signal that is being caught and a function defined by POSIX.1 {3} or by the C Standard {2} is called from the signal-catching function, then, unless both the interrupted function and the function called from the signal-catching function are marked as unsafe, the behavior is as defined in POSIX.1 {3}.

See Reason 1 in Section 5. of POSIX.3 {4}.

D11(C) If the action taken when a signal interrupts an unsafe function and the signal-catching function calls an unsafe function, and this is documented:
The details on the behavior of those functions is contained in 3.3.1.3 of the PCD.1.

3.3.1.4 Signal Effects on Other Functions

49(A) When a signal is delivered to a process during the execution of any function, and the action associated with the signal is to terminate the process, then the process is terminated, and the original function call does not return.

50(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a signal is delivered to a process during the execution of any function, and the action associated with the signal is to stop the process, then the process is stopped. A SIGCONT signal sent while the process is stopped causes the original function to continue from the point at which it was stopped.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

51(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a signal is delivered to a process during the execution of any function, and the action associated with the signal is to stop the process, then the process is stopped. A SIGKILL signal sent while the process is stopped causes the process to terminate.

52(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When any of the functions *fcntl*(), *open*(), *pause*(), *read*(), *sleep*(), *sigsuspend*(), *wait*(), *waitpid*(), and *write*() are stopped by the delivery of a signal whose action is to stop the process, then the subsequent generation of a SIGCONT signal that is not being caught by a signal-catching function causes the process to continue without the function reporting that it has been interrupted by either the stopping or the SIGCONT signal.

53(PCTS_GTI_DEVICE && PCTS_GTI_BUFFERS_OUTPUT?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the function *tcdrain*() is stopped by the delivery of a signal whose action is to stop the process, then the subsequent generation of a SIGCONT signal that is not being caught by a signal-catching function causes the process to continue without the *tcdrain*() function reporting that it has been interrupted by either the stopping or the SIGCONT signal.

54(D) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When any of the functions *_exit*(), *access*(), *alarm*(), *cfgetispeed*(), *cfgetospeed*(), *cfsetispeed*(), *cfsetospeed*(), *chdir*(), *chmod*(), *chown*(), *close*(), *creat*(), *dup*(), *dup2*(), *execle*(), *execve*(), *fork*(), *fstat*(), *getegid*(), *geteuid*(), *getgid*(), *getgroups*(), *getpgrp*(), *getpid*(), *getppid*(), *getuid*(), *kill*(), *link*(), *lseek*(), *mkdir*(), *inkfifo*(), *pathconf*(), *pipe*(), *rename*(), *rmdir*(), *setgid*(), *setpgid*(), *setaid*(), *setuid*(), *sigaction*(), *sigaddset*(), *sigdelset*(), *sigemptyset*(), *sigfillset*(), *sigpending*(), *sigprocmask*(), *sigsimember*(), *stat*(), *sysconf*(), *tcflow*(), *tcflush*(), *tcgetattr*(), *tcgetpgrp*(), *tcsendbreak*(), *tcsetattr*(), *tcsetpgrp*(), *time*(), *times*(), *umask*(), *uname*(), *unlink*(), *ustat*(), and *utime*() are stopped by the delivery of a signal whose action is to stop the process, then the subsequent generation of a SIGCONT signal that is not being caught by a signal-catching function causes the process to continue without the function reporting that it has been interrupted by either the stopping or the SIGCONT signal.

See Reason 3 in Section 5. of POSIX.3 [4].

55(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When any of the functions *fcntl*(), *open*(), *pause*(), *read*(), *sleep*(), *sigsuspend*(), *wait*(), *waitpid*(), and *write*() are stopped by the delivery of a signal whose action is to stop the process, then the subsequent generation of a SIGCONT signal that is being caught by a signal-catching function causes the process to continue after returning from the signal-catching function, with the original function reporting that it has been interrupted by the SIGCONT signal.

56(PCTS_GTI_DEVICE && PCTS_GTI_BUFFERS_OUTPUT?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the function *tcdrain*() is stopped by the delivery of a signal whose action is to stop the process, then the subsequent generation of a SIGCONT signal that is being caught by a signal-catching function causes the process to continue after returning from the signal-catching function, with the *tcdrain*() function reporting that it has been interrupted by the SIGCONT signal.

57(D) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When any of the functions *_exit*(), *access*(), *alarm*(), *cfgetispeed*(), *cfgetospeed*(), *cfsetispeed*(), *cfsetospeed*(), *chdir*(), *chmod*(), *chown*(), *close*(), *creat*(), *dup*(), *dup2*(), *execle*(), *execve*(), *fork*(), *fstat*(), *getegid*(), *geteuid*(), *getgid*(), *getgroups*(), *getpgrp*(), *getpid*(), *getppid*(), *getuid*(), *kill*(), *link*(), *lseek*(), *mkdir*(), *mkfifo*(), *pathconf*(), *pipe*(), *rename*(), *rmdir*(), *setgid*(), *setpgid*(), *setsid*(), *setuid*(), *sigaction*(), *sigaddset*(), *sigdelset*(), *sigemptyset*(), *sigfilleet*(), *sigpending*(), *sigprocmask*(), *sigsimember*(), *stat*(), *sysconf*(), *tcflow*(), *tcflush*(), *tcgetattr*(), *tcgetpgrp*(), *tcsendbreak*(), *tcsetattr*(), *tcsetpgrp*(), *time*(), *times*(), *umask*(), *uname*(), *unlink*(), *ustat*(), and *utime*() are stopped

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860204 562 ■

by the delivery of a signal whose action is to stop the process, then the subsequent generation of a SIGCONT signal that is being caught by a signal-catching function causes the process to continue after returning from the signal-catching function, with the original function reporting that it has been interrupted by the SIGCONT signal.

See Reason 3 in Section 5. of POSIX.3 {4}.

58(A) When any signal that is being ignored is generated while any function is being executed, then the signal does not have any effect on the behavior of the function.

Testing Requirements:

Test for the functions *fcntl*(), *open*(), *pause*(), *read*(), *sleep*(), *sigsuspend*(), *wait*(), *waitpid*(), and *write*(), each of which can be placed in a state where they would report being interrupted by a signal if it were delivered. The action of SIGALRM on *sleep*() is unspecified.

59(PCTS_GTI_DEVICE?A:UNTESTED)

When any signal that is being ignored is generated while *tcdrain*() is being executed, then the signal does not have any effect on the behavior of *tcdrain*().

60(A) When any signal that is being blocked is generated while any function is being executed, then the signal does not have any effect on the behavior of the function.

Testing Requirements:

Test for the functions *fcntl*(), *open*(), *pause*(), *read*(), *sleep*(), *sigsuspend*(), *wait*(), *waitpid*(), and *write*(), each of which can be placed in a state where they would report being interrupted by a signal if it were delivered. The action of SIGALRM on *sleep*() is unspecified.

61(PCTS_GTI_DEVICE?A:UNTESTED)

When any signal that is being blocked is generated while *tcdrain*() is being executed, then the signal does not have any effect on the behavior of *tcdrain*().

3.3.2 *kill*()

3.3.2.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header `<signal.h>` is included, then the function prototype `int kill(pid_t, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header `<signal.h>` is included, then the function *kill*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *kill*() is defined as a macro when the header `<signal.h>` is included:

When the macro *kill*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *kill*() is defined as a macro in the header `<signal.h>`:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.3.2.2 Description

04(A) When a process has the permissions allowing it to send a signal to the process(es) specified by *pid*, then a call to *kill*(*pid*, *sig*) can send any of the signals SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, and SIGUSR2. On successful generation of the signal, the return value is zero.

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

05(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a process has the permissions allowing it to send a signal to the process(es) specified by *pid*, then a call to *kill(pid, sig)* can send any of the signals SIGCHLD, SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, and SIGCONT. On successful generation of the signal, the return value is zero.

06(A) When a process has permission allowing it to send the null signal to the process specified by *pid*, then a call to *kill(pid, 0)* returns a value of zero.

07(A) When *pid* is greater than 0 and the real user ID of the sending process matches the real user ID of the receiving process, then a ca6ll of *kill(pid, sig)* sends the signal *sig* to the process *pid*.

08(A) When *pid* is greater than 0 and the effective user ID of the sending process matches the real user ID of the receiving process, then a call of *kill(pid, sig)* sends the signal *sig* to the process *pid*.

09(A) If the behavior associated with {_POSIX_SAVED_IDS} is supported:
When *pid* is greater than 0 and the real user ID of the sending process matches the saved set-user ID of the receiving process, then a call of *kill(pid, sig)* sends the signal *sig* to the process *pid*.

Otherwise:
When *pid* is greater than 0 and the real user ID of the sending process matches the effective user ID of the receiving process, then a call of *kill(pid, sig)* sends the signal *sig* to the process *pid*.

10(A) If the behavior associated with {_POSIX_SAVED_IDS} is supported:
When *pid* is greater than 0 and the effective user ID of the sending process matches the saved set-user ID of the receiving process, then a call of *kill(pid, sig)* sends the signal *sig* to the, process *pid*.

Otherwise:
When *pid* is greater than 0 and the effective user ID of the sending process matches the effective user ID of the receiving process, then a call of *kill(pid, sig)* sends the signal *sig* to the process *pid*.

D01(C) If the implementation provides a mechanism to associate with a process the appropriate privileges to send signals to processes not associated with the current UID:
The details are documented in 2.2.2.4 or 3.3.2.2 of the PCD.

11(C) If the implementation provides the appropriate privileges to signal any other process:
When a process has the appropriate privileges to send signals to a process *pid* that has different real, effective, or saved set-user IDs, then a call to *kill(pid, sig)* with *pid* greater than 0 sends the signal *sig* to the process *pid*.

D02(C) If the implementation excludes a set of system processes from being sent a signal *sig* when *pid* is 0, and this is documented:
The set of system processes that are excluded is contained in 3.3.2.2 of the PCD.1.

12(A) A call to *kill((pid_t)0, sig)* from a process sends the signal *sig* to all processes (excluding the set of system processes defined by the implementation as not receiving the signal) whose process group ID is equal to the process group ID of the sender and for which the process has permission to send the signal.

D03(C) If the behavior of the *kill*() function when *pid* is −1 is documented:
The details are contained in 3.3.2.2 of the PCD.1.

D04(C) If the implementation excludes a set of system processes from being sent a signal *sig* when *pid* is negative, but not -1, and this is documented:
The set of system processes that are excluded is contained in 3.3.2.2 of the PCD.1.

13(A) When *pid* is negative but not −1, a call to *kill(pid, sig)* causes the signal *sig* to be sent to all process whose process group ID is equal to the absolute value of *pid* and for which the process has permission to send the signal.

14(A) When a call to *kill(pid, sig)* uses a value of *pid* that causes a signal to be sent to the calling process, and when *sig* is not blocked by the calling process, then either *sig* or at least one pending unblocked signal is delivered to the sending process before the *kill*() function returns.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860206 335 ■

15(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process specified by *pid* is a member of the same session as the calling process, then a call to *kill(pid, SIGCONT)* sends the SIGCONT signal to the process specified by *pid* irrespective of its user ID.

D05(C) If extended security control imposes restrictions on the sending of signals not defined by POSIX.1 {3}:
The restrictions imposed on the sending of signals, including the NULL signal, are contained in 3.3.2.2 of the PCD.1.

3.3.2.3 Returns

R01 When a call to *kill*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 4–6 in 3.3.2.2.)

R02 When a call to *kill*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertions 16–20 in 3.3.2.4.)

R03 When a call to *kill*() completes unsuccessfully, no signal is sent. (See Assertions 16–20 in 3.3.2.4.)

3.3.2.4 Errors

16(PCTS_INVALID_signal?A:UNTESTED)

When the value of *sig* is an invalid or an unsupported signal number, then a call to *kill(pid, sig)* returns a value of (*int*)–1, sets *errno* to [EINVAL], and the signal is not sent.

17(A) If the behavior associated with {_POSIX_SAVED_IDS} is supported:
When the process does not have the appropriate privileges to send signals to the process of another user, and when the real or effective user ID of the sending process does not match the real or saved setuser ID of any receiving process, and when *sig* is not equal to SIGCONT, then a call to *kill(pid, sig)* returns a value of (*int*)-1, sets *errno* to [EPERM], and the signal is not sent.

Otherwise:
When the sending process does not have the appropriate privileges, and when the real or effective user ID of the sending process does not match the effective user ID of any receiving process, and when *sig* is not equal to SIGCONT, then a call to *kill(pid, sig)* returns a value of (*int*)-1, sets *errno* to [EPERM], and the signal is not sent.

18(C) If the behaviors associated with {_POSIX_JOB_CONTROL} and {_POSIX_SAVED_IDS} are supported:
When the process does not have the appropriate privileges to send signals to the process of another user, and when the real or effective user ID of the sending process does not match the real or saved setuser ID of any receiving process, and when there are no receiving processes in the same session as the sending process and argument *sig* is equal to SIGCONT, then a call to *kill(pid, sig)* returns a value of (*int*)-1, sets *errno* to [EPERM], and the SIGCONT signal is not sent.

19(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported and the behavior associated with {_POSIX_SAVED_IDS} is not supported:
When the process does not have the appropriate privileges to send signals to the process of another user, and when the real or effective user ID of the sending process does not match the effective user ID of any receiving process, and when there are no receiving processes in the same session as the sending process and argument *sig* is equal to SIGCONT, then a call to *kill(pid, sig)* returns a value of (*int*)-1, sets *errno* to [EPERM], and the SIGCONT signal is not sent.

20(A) When no process and no process group can be found corresponding to that specified by *pid*, then a call to *kill(pid, sig)* returns a value of (*int*)-1, sets *errno* to [ESRCH], and the signal is not sent.

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.3.3 Manipulate Signal Sets

3.3.3.1 *sigemptyset*() (3.3.3)

3.3.3.1.1 Synopsis (3.3.3.1)

01(A) If the implementation provides C Standard {2} support:
When the header <signal.h> is included, then the function prototype `int sigemptyset(sigset_t *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <signal.h> is included, then the function *sigemptyset*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *sigemptyset*() is defined as a macro when the header <signal.h> is included:
When the macro *sigemptyset*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sigemptyset*() is defined as a macro in the header <signal.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.3.3.1.2 Description (3.3.3.2)

04(A) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
A call to *sigemptyset(set)* initializes the signal set pointed to by the argument set such that the signals SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU are excluded and a value of zero is returned.

Otherwise:
A call to *sigemptyset(set)* initializes the signal set pointed to by the argument set such that the signals SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, and SIGUSR2 are excluded and a value of zero is returned.

D01(C) If the action taken when an object of type *sigset_t* has not been initialized by a call to *sigemptyset*() or *sigfillset*(), but a pointer to the object is supplied as an argument to the functions *sigaddset*(), *sigdelset*(), *sigismember*(), *sigaction*(), *sigrocmask*(), *sigpending*(), or *sigsuspend*(), and this is documented:
The details are contained in 3.3.3.2 of the PCD.1.

3.3.3.1.3 Returns (3.3.3.3)

R01 When a call to *sigemptyset*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 4 in 3.3.3.1.2.)

3.3.3.1.4 Errors (3.3.3.4)

There are no assertions specific to this subclause.

3.3.3.2 *sigfillset*() (3.3.3)

3.3.3.2.1 Synopsis (3.3.3.1)

01(A) If the implementation provides C Standard {2} support:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the header <signal.h> is included, then the function prototype
`int sigfillset(sigset_t *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <signal.h> is included, then the function *sigfillset*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *sigfillset*() is defined as a macro when the header <signal.h> is included:

When the macro *sigfillset*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sigfillset*() is defined as a macro in the header <signal.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.3.3.2.2 Description (3.3.3.2)

04(A) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

A call to *sigfillset*(*set*) initializes the signal set pointed to by the argument *set* such that the signals SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU are included and a value of zero is returned.

Otherwise:

A call to *sigfillset*(*set*) initializes the signal set pointed to by the argument *set* such that the signals SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, and SIGUSR2 are included and a value of zero is returned.

3.3.3.2.3 Returns (3.3.3.3)

R01 When a call to *sigfillset*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 4 in 3.3.3.2.2.)

3.3.3.2.4 Errors (3.3.3.4)

There are no assertions specific to this subclause.

3.3.3.3 *sigaddset*() (3.3.3)

3.3.3.3.1 Synopsis (3.3.3.1)

01(A) If the implementation provides C Standard {2} support:

When the header <signal.h> is included, then the function prototype
`int sigaddset(sigset_t *, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <signal.h> is included, then the function *sigaddset*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *sigaddset*() is defined as a macro when the header <signal.h> is included:

When the macro *sigaddset*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sigaddset*() is defined as a macro in the header <signal.h>:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.3.3.3.2 Description (3.3.3.2)

04(A) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

A call to *sigaddset(set, signo)* where the value of the *signo* argument is SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU adds the *signo* argument to the signal set pointed to by the argument *set*, and a value of zero is returned.

Otherwise:

A call to *sigaddset(set, signo)* where the value of the *signo* argument is SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, or SIGUSR2 adds the *signo* argument to the signal set pointed to by the argument *set*, and a value of zero is returned.

3.3.3.3.3 Returns (3.3.3.3)

R01 When a call to *sigaddset*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 4 in 3.3.3.3.2.)

R02 When a call to *sigaddset*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error. (See Assertion 5 in 3.3.3.3.4.)

3.3.3.3.4 Errors (3.3.3.4)

D01(C) If the implementation supports the detection of [EINVAL] for *sigaddset*():

The details under which [EINVAL] occurs for *sigaddset*() are contained in 3.3.3.4 of the PCD.1. (See DGA02 in 2.4.)

05(PCTS_INVALID_SIGNAL?C:UNTESTED)

If the implementation supports the detection of [EINVAL] for *sigaddset*():

When the value of the argument *signo* is an invalid or unsupported signal number, then a call to *sigaddset(set, signo)* returns a value of (*int*)-1 and sets *errno* to [EINVAL]. The signal set pointed to by the argument *set* is not changed.

06(D) If the implementation does not support the detection of [EINVAL] for *sigaddset*():

A call to *sigaddset(set, signo)* is successful for all possible values of *signo* (unless a different error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 {4}.

3.3.3.4 *sigdelset*() (3.3.3)

3.3.3.4.1 Synopsis (3.3.3.1)

01(A) If the implementation provides C Standard {2} support:

When the header <signal.h> is included, then the function prototype `int sigdelset(sigset_t *, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <signal.h> is included, then the function *sigdelset*() is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *sigdelset*() is defined as a macro when the header <signal.h> is included:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the macro *sigdelset*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sigdelset*() is defined as a macro in the header <signal.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.3.3.4.2 Description (3.3.3.2)

04(A) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
A call to *sigdelset(set, signo)* where the value of the *signo* argument is SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU deletes the *signo* argument from the signal set pointed to by the argument *set*, and a value of zero is returned.

Otherwise:
A call to *sigdelset(set, signo)* where the value of the *signo* argument is SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, or SIGUSR2 deletes the *signo* argument from the signal set pointed to by the argument *set*, and a value of zero is returned.

3.3.3.4.3 Returns (3.3.3.3)

R01 When a call to *sigdelset*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 4 in 3.3.3.4.2.)

R02 When a call to *sigdelset*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertion 5 in 3.3.3.4.4.)

3.3.3.4.4 Errors (3.3.3.4)

D01(C) If the implementation supports the detection of [EINVAL] for *sigdelset*():
The details under which [EINVAL] occurs for *sigdelset*() are contained in 3.3.3.4 of the PCD.1. (See DGA02 in 2.4.)

05(PCTS_INVALID_SIGNAL?C:UNTESTED)

If the implementation supports the detection of [EINVAL] for *sigdelset*():
When the value of the argument *signo* is an invalid or unsupported signal number, then a call to *sigdelset(set, signo)* returns a value of (*int*)-1 and sets *errno* to [EINVAL]. The signal set pointed to by the argument *set* is not changed.

06(D) If the implementation does not support the detection of [EINVAL] for *sigdelset*():
A call to *sigdelset(set, signo)* is successful for all possible values of *signo* (unless a different error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 {4}.

3.3.3.5 *sigismember*() (3.3.3)

3.3.3.5.1 Synopsis (3.3.3.1)

01(A) If the implementation provides C Standard {2} support:
When the header <signal.h> is included, then the function prototype
`int sigismember(const sigset_t *, int)` is declared. (See GA36 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Otherwise:

When the header <signal.h> is included, then the function *sigismember*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *sigismember*() is defined as a macro when the header <signal.h> is included:

When the macro *sigismember*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sigismember*() is defined as a macro in the header <signal.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.3.3.5.2 Description (3.3.3.2)

04(A) If the behavior associated with {_POSIX JOB_CONTROL} is supported:

A call to *sigismember(set, signo)* where the value of the *signo* argument is SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU tests whether the *signo* argument is a member of the signal set pointed to by the argument *set* and returns a value of 1 if the specified signal is a member of the set and a value of 0 if it is not.

Otherwise:

A call to *sigismember(set, signo)* where the value of the *signo* argument is SIGABRT, SIGALRM, SIGFPE, SIGHUP, SIGILL, SIGINT, SIGKILL, SIGPIPE, SIGQUIT, SIGSEGV, SIGTERM, SIGUSR1, or SIGUSR2 tests whether the *signo* argument is a member or the signal set pointed to by the argument *set* and returns a value of 1 if the specified signal is a member of the set and a value of 0 if it is not.

3.3.3.5.3 Returns (3.3.3.3)

R01 When the value of the *signo* argument is a member of *set*, then a call to *sigismember(set, signo)* returns a value of (*int*)1. (See Assertion 4 in 3.3.3.5.2.)

R02 When the value of the *signo* argument is not a member of *set*, then a call to *sigismember(set, signo)* returns a value of (*int*)0. (See Assertion 4 in 3.3.3.5.2.)

R03 When a call to *sigismember*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error. (See Assertion 5 in 3.3.3.5.4.)

3.3.3.5.4 Errors (3.3.3.4)

D01(C) If the implementation supports the detection of [EINVAL] for *sigismember*():

The details under which [EINVAL] occurs for *sigismember*() are contained in 3.3.3.4 of the PCD.1. (See DGA02 in 2.4.)

05(PCTS INVALID_SIGNAL?C:UNTESTED)

If the implementation supports the detection of [EINVAL] for *sigismember*():

When the value of the argument *signo* is an invalid or unsupported signal number, then a call to *sigismember(set, signo)* returns a value of (*int*)-1 and sets *errno* to [EINVAL].

06(D) If the implementation does not support the detection of [EINVAL] for *sigismember*():

A call to *sigismember(set, signo)* is successful for all possible values of *signo* (unless a different error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860212 639 ■

3.3.4 *sigaction* ()

3.3.4.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <signal.h> is included, then the function prototype `int sigaction(int, const struct sigaction *, struct sigaction *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <signal.h> is included, then the function *sigaction*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *sigaction*() is defined as a macro when the header <signal.h> is included:

When the macro *sigaction*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sigaction*() is defined as a macro in the header <signal.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.3.4.2 Description

04(A) When the header <signal.h> is included, then the structure *sigaction* is defined, and this structure defines the following members, as shown in Table 3.1.

Table 3.1—Structure *sigaction* Members

Member	Type
sa_handler	*void(*)()*
sa_mask	*sigset_t*
sa_flags	*int*

05(A) A call to *sigaction(sig, NULL, oact)* does not alter the signal handling, places the action currently associated with the signal in the location pointed to by the non-NULL argument *oact*, and returns a value of zero.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not affect the behavior of the call to *sigaction*().

06(A) A call to *sigaction(sig, act, NULL)* associates the action specified by a non-NULL *act* with the signal *sig* and returns a value of zero.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

07(A) A call to *sigaction(sig, act, oact)* associates the action specified by a non-NULL *act* with the signal *sig* and stores the action previously associated with the signal in the location pointed to by the non-NULL argument *oact*.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

08(A) The *sa_handler* field of the *sigaction* structure can be set to SIG_DFL, SIG_IGN, or to a signal-handling function.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

09(A) When a signal is delivered, then the action defined in the *sa_handler* field of the *sigaction* structure associated with the signal at the time of delivery is performed.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3}; and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

10(A) When a signal is caught by a signal-catching function installed by *sigaction*(), then a new signal mask is calculated by taking the union of the current signal mask, the *sigaction* signal mask for the signal being delivered, and the signal being delivered. The signals SIGSTOP and SIGKILL are not included in the new signal mask. The new signal mask is installed before entering the signal-catching function and remains in effect until

— The signal catching function returns,
— The function *sigprocmask*() is called, or
— The function *sigsuspend*() is called.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

11(A) When the signal handler for the user returns normally, then the original signal mask is restored.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

12(A) When the header `<signal.h>` is included, then the flag SA_NOCLDSTOP is defined and can be set in *sa_flags*.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

13(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When *sig* is set to SIGCHLD and the SA_NOCLDSTOP flag is clear in *sa_flags*, then a call to *sigaction*(*sig, act, oact*) causes a SIGCHLD signal to be generated for the parent process whenever any of its child processes stop.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

14(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When *sig* is SIGCHLD and the SA_NOCLDSTOP flag is set in *sa_flags*, then a call to *sigaction*(*sig, act, oact*) results in a SIGCHLD signal not being generated for the parent process whenever any of its child processes stop.

Testing Requirements:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

15(A) When an action is installed for a specific signal, then it remains installed until another action is explicitly requested by another call to *sigaction*() or until one of the `exec`() functions is called.

Testing Requirements:

Testing the change of signal action associated with a call to `exec`() need not be tested as part of this assertion; it is more thoroughly covered in other places.

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

D01(C) If the action taken when *sigaction*() is called, when the previous action for signal *sig* had been established by the *signal*() function as defined in the C Standard {2}, is documented:

The details on the values of the fields returned in the structure pointed to by *oact* for a call to *sigaction*() are contained in 3.3.4.2 of the PCD.1.

16(C) If the function *signal*() is supported:

When a signal-handling function has been installed using the *signal*() function, then the value supplied in the *oact* structure member *sa_handler* on a call to *sigaction*() can be used by *sigaction*() to reestablish the original signal-handling function.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3}; and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

R01 When *sigaction*() fails, then no new signal handler is installed. (See Assertions 20 and 21 in 3.3.4.4.)

D02(C) If the action taken when an attempt to set the action for a signal that cannot be caught or ignored to SIG_DFL is made, and this is documented:

The details on whether the action is ignored or causes an error to be returned are contained in 3.3.4.2 of the PCD.1.

17(A) When *sigaction*(*sig, act, oact*) is called to set the action for SIGKILL to SIG_DFL, then either a value of -1 is returned and *errno* is set to [EINVAL] or a value of 0 is returned.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

18(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When *sigaction*(*sig, act, oact*) is called to set the action for SIGSTOP to SIG_DFL, then either a value of -1 is returned and *errno* is set to [EINVAL] or a value of 0 is returned.

Testing Requirements:

When the *sigaction* structure contains additional members beyond those specified in POSIX.1 {3} and the effect of these members is not activated, then these additional structure members do not effect the behavior of the call to *sigaction*().

3.3.4.3 Returns

R02 When a call to *sigaction*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 5 and 6 in 3.3.4.2.)

R03 When a call to *sigaction*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertions 19-21 in 3.3.4.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.3.4.4 Errors

19(PCTS_INVALID_SIGNAL?A:UNTESTED)

When the value of the *sig* argument is an invalid or unsupported signal number, then a call to *sigaction(sig, act, oact)* returns a value of (*int*)-1 and sets *errno* to [EINVAL].

20(A) A call to *sigaction(sig, act, oact)* to catch or ignore SIGKILL returns a value of (*int*)-1, sets *errno* to [EINVAL], and does not install a new handler.

21(C) If the behavior associated with SIGSTOP is supported:

A call to *sigaction(sig, act, oact)* to catch or ignore SIGSTOP returns a value of (*int*)-1, sets *errno* to [EINVAL], and no new handler is installed.

R04 An attempt to set the action to SIG_DFL for signal SIGKILL or SIGSTOP either returns a value of [(*int*)-1] and sets *errno* to [EINVAL], or returns a value of [(*int*)0]. (See Assertions 17 and 18 in 3.3.4.2.)

3.3.5 *sigprocmask*()

3.3.5.1 Synopsis

01(A) If the implementation provides C Standard [2] support:

When the header `<signal.h>` is included, then the function prototype `int sigprocmask(int,const sigset_t *, sigset_t *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header `<signal.h>` is included, then the function *sigprocmask*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *sigprocmask*() is defined as a macro when the header `<signal.h>` is included:

When the macro *sigprocmask*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sigprocmask*() is defined as a macro in the header `<signal.h>`:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.3.5.2 Description

04(A) When the header `<signal.h>` is included, then the symbols SIG_BLOCK, SIG_UNBLOCK, and SIG_SETMASK are defined.

05(A) A call to *sigprocmask(SIG_BLOCK, set, oset)* changes the signals blocked for the process to the union of the current blocked signals and the signal set pointed to by the non-NULL argument *set*, and returns zero. When *oset* is non-NULL, the previous mask is stored in the location pointed to by *oset*.

06(A) A call to *sigprocmask(SIG_UNBLOCK, set, oset)* changes the signals blocked for the process to the intersection of the current blocked signals and the complement of the signal set pointed to by the non-NULL argument *set*, and returns zero. When *oset* is non-NULL, the previous mask is stored in the location pointed to by *oset*.

07(A) A call to *sigprocmask(SIG_SETMASK, set, oset)* changes the signals blocked for the process to the signal set pointed to by the non-NULL argument *set* and returns zero. When *oset* is non-NULL, the previous mask is stored in the location pointed to by *oset*.

08(A) A call to *sigprocmask(how, NULL, oset)* does not change the signal mask of the process. When *oset* is non-NULL, the previous mask is stored in the location pointed to by *oset*.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860216 284 ■

09(A) A call to *sigprocmask(how, set, NULL)* changes the signals blocked for the process according to the directive *how* and the signal set pointed to by the non-NULL argument *set*.

10(A) A call to *sigprocmask(how, NULL, NULL)* does not change the signal mask of the process.

11(A) When SIGKILL is included in the set pointed to by the set argument, then the *sigprocmask*() call succeeds, and SIGKILL is not blocked from delivery.

12(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When SIGSTOP is included in the set argument, then the *sigprocmask*() call succeeds, and SIGSTOP is not blocked from delivery.

D01(C) If the action taken when any of the signals SIGFPE, SIGILL, or SIGSEGV are generated while blocked, and the signal was not generated by a call to *kill*() or *raise*(), and this is documented:
The details on the action to be taken by the implementation are contained in 3.3.5.2 of the PCD.1.

R01 When the *sigprocmask*() function fails, then the signal mask of the process is not changed. (See Assertion 14 in 3.3.5.4.)

13(A) When there are any pending unblocked signals after the call to the *sigprocmask*() function, then at least one of those signals is delivered before the *sigprocmask*() function returns.

3.3.5.3 Returns

R02 When a call to *sigprocmask*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 5-7 in 3.3.5.2.)

R03 When a call to *sigprocmask*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertion 14 in 3.3.5.4.)

3.3.5.4 Errors

14(A) When the value of the *how* argument is not one of the defined values and *set* is non-NULL, then a call to *sigprocmask(how, set, oset)* returns a value of (*int*)−1, sets *errno* to [EINVAL], and does not change the signal mask of the process.

3.3.6 *sigpending*()

3.3.6.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <signal.h> is included, then the function prototype `int sigpending(sigset_t *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <signal.h> is included, then the function *sigpending*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

002(C) If *sigpending*() is defined as a macro when the header <signal.h> is included:
When the macro *sigpending*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sigpending*() is defined as a macro in the header <signal.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.3.6.2 Description

04(A) A call to *sigpending(set)* stores the set of signals that are pending and blocked from delivery in the calling process in the space pointed to by the argument set, and returns a value of zero.

3.3.6.3 Returns

R01 When a call to *sigpending*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 4 in 3.3.6.2.)

05(D) If the implementation supports error conditions for *sigpending*():
When a call to *sigpending*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error.

See Reason 2 in Section 5. of POSIX.3 {4}.

3.3.6.4 Errors

D01(C) If the implementation supports error conditions for *sigpending*(), and this is documented:
The conditions under which errors are detected for *sigpending*() is stated in 3.3.6.4 of the PCD.1. (See DGA02 in 2.4.)

3.3.7 *sigsuspend*()

3.3.7.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <signal.h> is included, then the function prototype
`int sigsuspend(const sigset_t *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <signal.h> is included, then the function *sigsuspend*() is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *sigsuspend*() is defined as a macro when the header <signal.h> is included:
When the macro *sigsuspend*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sigsuspend*() is defined as a macro in the header <signal.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.3.7.2 Description

04(A) A call to *sigsuspend(sigmask)* replaces the signal mask of the calling process with the set of signals pointed to by the argument sigmask and then suspends the process until delivery of a signal whose action is to execute a signal-catching function or to terminate the process.

05(A) When the action of the signal delivered to the suspended by a call to *sigsuspend*() is to terminate the process, then the *sigsuspend*() function does not return.

06(A) When the action of the signal delivered to the process suspended by a call to *sigsuspend*() is to execute a signal-catching function, then on return *sigsuspend*() restores the signal mask to the set that existed prior to the *sigsuspend*() call.

07(A) A call to *sigsuspend*() attempting to block SIGKILL will cause no error, and the receipt of the SIGKILL will result in the default action for that signal.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

08(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
A call to *sigsuspend*() attempting to block SIGSTOP will cause no error, and the receipt of SIGSTOP will result in the default action for that signal.

3.3.7.3 Returns

R01 A call to *sigsuspend*() is always unsuccessful. It returns a value of [(*int*)-1] and sets *errno* to indicate the error. (See Assertion 9 in 3.3.7.4.)

3.3.7.4 Errors

09(A) When a signal is caught by the calling process and control is returned from the signal catching function, then a call to *sigsuspend*() returns a value of (*int*)-1 and sets *errno* to [EINTR].

3.4 Timer Operations

3.4.1 *alarm*()

3.4.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `unsigned int alarm(unsigned int)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *alarm*() is declared with the result type *unsigned int*. (See GA36 in 2.7.3.)

02(C) If *alarm*() is defined as a macro when the header <unistd.h> is included:
When the macro *alarm*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *unsigned int*. (See GA37 in 2.7.3.)

03(C) If *alarm*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.4.1.2 Description

04(A) When the value of the argument seconds is nonzero, then a call to *alarm*(*seconds*) causes the system to send the calling process a SIGALRM signal after the number of real-time seconds specified by the argument *seconds* have elapsed.

05(A) Successive calls to *alarm*() reschedule the alarm clock for the calling process.

06(A) A call to *alarm*(0) cancels any previously made *alarm*() request.

3.4.1.3 Returns

07(A) When there is a previous *alarm*() request with at least 1 s remaining, the return value of *alarm*() is the amount of time remaining in seconds before the system is scheduled to generate the SIGALRM signal.

08(A) When there is no previous *alarm*() request, the return value of *alarm*() is zero.

09(A) When a call to *alarm*() returns and there is a previous *alarm*() request with less than 1 s remaining, then the return value is 1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.4.1.4 Errors

There are no assertions specific to this subclause.

3.4.2 *pause*()

3.4.2.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `inc pause(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *pause*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *pause*() is defined as a macro when the header <unistd.h> is included:
When the macro *pause*() is invoked, then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *pause*() is defined as a macro in the header <unistd.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.4.2.2 Description

04(A) A call to *pause*() suspends the calling process until the delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

05(A) When a signal is delivered and the action is to terminate the process, then *pause*() does not return.

06(A) When a signal is delivered and the signal action is to execute a signal-catching function, then *pause*() returns after the signal-catching function returns.

3.4.2.3 Returns

R01 A call to *pause*() is always unsuccessful. It returns a value of (*int*)-1 and sets *errno* to [EINTR]. (See Assertion 7 in 3.4.2.4.)

3.4.2.4 Errors

07(A) When a signal is caught by the calling process and control is returned from the signal-catching function, then a call to *pause*() returns a value of (*int*)-1 and sets *errno* to [EINTR].

3.4.3 *sleep*()

3.4.3.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `unsigned int sleep(unsigned int)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *sleep*() is declared with the result type *unsigned int*. (See GA36 in 2.7.3.)

02(C) If *sleep*() is defined as a macro when the header <unistd.h> is included:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the macro *sleep*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *unsigned int*. (See GA37 in 2.7.3.)

03(C) If *sleep*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

3.4.3.2 Description

04(A) When the number of real-time seconds specified by the value of the argument *seconds* have elapsed since a call of *sleep*(*seconds*) was made, then the call to *sleep*(*seconds*) returns a value of zero.

05(A) When a signal is delivered before the number of real-time seconds specified by the value of the argument *seconds* have elapsed, and when the action associated with the signal is to invoke a signal-catching function, and when this signal-catching function does not alter the disposition or the action associated with the SIGALRM function, then on return from the signal-catching function the call to *sleep*(*seconds*) returns the unslept time in seconds.

06(A) When a signal is delivered before the number of real-time seconds specified by the value of the argument *seconds* have elapsed, and when the action associated with this signal is to terminate the process, then the call to *sleep*(*seconds*) does not return.

D01(C) If the action taken, when the SIGALRM signal generated for the calling process during execution of the *sleep*() function is being ignored or blocked from delivery, is documented:
The details on whether *sleep*() returns when the SIGALRM signal is scheduled are contained in 3.4.3.2 of the PCD.1.

D02(C) If the action taken, when the SIGALRM signal is blocked from delivery, is documented:
The details on whether SIGALRM is discarded or remains pending after the *sleep*() function returns are contained in 3.4.3.2 of the PCD.1.

D03(C) If the action taken, when the SIGALRM signal generated for the calling process during execution of the *sleep*() function—except as a result of a prior call to *alarm*()—is not being ignored or blocked from delivery, is documented:
The details on whether signal SIGALRM has any effect other than causing the *sleep*() function to return is contained in 3.4.3.2 of the PCD.1.

D04(C) If the action taken with the SIGALRM signal—whether the SIGALRM signal is blocked from delivery—when a signal-catching function interrupts the *sleep*() function and either examines or changes the time a SIGALRM is scheduled to be delivered, is documented:
The time the SIGALRM signal is to be delivered, the action associated with the SIGALRM signal, and whether the SIGALRM signal is blocked from delivery are contained in 3.4.3.2 of the PCD.1.

D05(C) If the action taken, when a signal-catching function interrupts *sleep*() and calls *siglongjmp*() or *longjmp*() to restore art environment saved prior to the *sleep*() call, is documented:
The details on the action associated with the SIGALRM signal, the time at which a SIGALRM signal is scheduled to be generated, and the action associated when the SIGALRM signal is blocked and the signal mask of the process is not restored as part of the environment are contained in 3.4.3.2 of the PCD.1.

3.4.3.3 Returns

R01 When a call to *sleep*() returns because the requested time has elapsed, a value of zero is returned. (See Assertion 4 in 3.4.3.2.)

R02 When interrupted by an alarm or another signal, then a call to *sleep*() returns the unslept time in seconds. (See Assertion 5 in 3.4.3.2.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.4.3.4 Errors

There are no assertions specific to this subclause.

4. Process Environment

4.1 Process Identification

4.1.1 Get Process and Parent Process IDs

4.1.1.1 *getpid*() (4.1.1)

4.1.1.1.1 Synopsis (4.1.1.1)

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `pid_t getpid(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *getpid*() is declared with the result type *pid_t*. (See GA36 in 2.7.3.)

02(C) If *getpid*() is defined as a macro when the header <unistd.h> is included:
When the macro *getpid*() is invoked, then it expands to an expression with the result type *pid_t*. (See GA37 in 2.7.3.)

03(C) If *getpid*() is defined as a macro in the header <unistd.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.1.1.1.2 Description (4.1.1.2)

04(A) A call to *getpid*() returns the process ID of the calling process.

4.1.1.1.3 Returns (4.1.1.3)

There are no assertions specific to this subclause.

4.1.1.1.4 Errors (4.1.1.4)

There are no assertions specific to this subclause.

4.1.1.2 *getppid*() (4.1.1)

4.1.1.2.1 Synopsis (4.1.1.1)

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `pid_t getppid(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *getppid*() is declared with the result type *pid_t*. (See GA36 in 2.7.3.)

02(C) If *getppid*() is defined as a macro when the header <unistd.h> is included:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the macro *getppid*() is invoked, then it expands to an expression with the result type *pid_t*. (See GA37 in 2.7.3.)

03(C) If *getppid*() is defined as a macro in the header <unistd.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.1.1.2.2 Description (4.1.1.2)

04(A) A call to *getppid()* returns the parent process ID of the calling process.

4.1.1.2.3 Returns (4.1.1.3)

There are no assertions specific to this subclause.

4.1.1.2.4 Errors (4.1.1.4)

There are no assertions specific to this subclause.

4.2 User Identification

4.2.1 Get Real User, Effective User, Real Group, and Effective Group IDs

4.2.1.1 *getuid*() (4.2.1)

4.2.1.1.1 Synopsis (4.2.1.1)

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `uid_t getuid(void)` is declared. (see GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *getuid*() is declared with the result type *uid_t*. (see GA36 in 2.7.3.)

02(C) If *getuid*() is defined as a macro when the header <unistd.h> is included:
When the macro *getuid*() is invoked, then it expands to an expression with the result type *uid_t*. (see GA37 in 2.7.3.)

03(C) If *getuid*() is defined as a macro in the header <unistd.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.2.1.1.2 Description (4.2.1.2)

04(A) A call to *getuid*() returns the real user ID of the calling process.

4.2.1.1.3 Returns (4.2.1.3)

There are no assertions specific to this subclause.

4.2.1.1.4 Errors (4.2.1.4)

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.2.1.2 *geteuid*() (4.2.1)

4.2.1.2.1 Synopsis (4.2.1.1)

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `uid_t geteuid(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *geteuid*() is declared with the result type *uid_t*. (See GA36 in 2.7.3.)

02(C) If *geteuid*() is defined as a macro when the header <unistd.h> is included:
When the macro *geteuid*() is invoked, then it expands to an expression with the result type *uid_t*. (See GA37 in 2.7.3.)

03(C) If *geteuid*() is defined as a macro in the header <unistd.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.2.1.2.2 Description (4.2.1.2)

04(A) A call to *geteuid*() returns the effective user ID of the calling process.

4.2.1.2.3 Returns (4.2.1.3)

There are no assertions specific to this subclause.

4.2.1.2.4 Errors (4.2.1.4)

There are no assertions specific to this subclause.

4.2.1.3 *getgid*() (4.2.1)

4.2.1.3.1 Synopsis (4.2.1.1)

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `gid_t getgid(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *getgid*() is declared with the result type *gid_t*. (See GA36 in 2.7.3.)

02(C) If *getgid*() is defined as a macro when the header <unistd.h> is included:
When the macro *getgid*() is invoked, then it expands to an expression with the result type *gid_t*. (See GA37 in 2.7.3.)

03(C) If *getgid*() is defined as a macro in the header <unistd.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.2.1.3.2 Description (4.2.1.2)

04(A) A call to *getgid*() returns the real group ID of the calling process.

4.2.1.3.3 Returns (4.2.1.3)

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860224 350 ■

4.2.1.3.4 Errors (4.2.1.4)

There are no assertions specific to this subclause.

4.2.1.4 *getegid*() (4.2.1)

4.2.1.4.1 Synopsis (4.2.1.1)

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `gid_t getegid(void)` is declared. (see GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *getegid*() is declared with the result type *gid_t*. (See GA36 in 2.7.3.)

02(C) If *getegid*() is defined as a macro when the header <unistd.h> is included:
When the macro *getegid*() is invoked, then it expands to an expression with the result type *gid_t*. (See GA37 in 2.7.3.)

03(C) If *getegid*() is defined as a macro in the header <unistd.h>:
It protects its return value with extra parentheses when necessary.(See GA01 in 1.3.4.)

4.2.1.4.2 Description (4.2.1.2)

04(A) A call to *getegid*() returns the effective group ID of the calling process.

4.2.1.4.3 Returns (4.2.1.3)

There are no assertions specific to this subclause.

4.2.1.4.4 Errors (4.2.1.4)

There are no assertions specific to this subclause.

4.2.2 Set User and Group IDs

4.2.2.1 *setuid*() (4.2.2)

4.2.2.1.1 Synopsis (4.2.2.1)

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `int setuid(uid_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *setuid*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *setuid*() is defined as a macro when the header <unistd.h> is included:
When the macro *setuid*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *setuid*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.2.2.1.2 Description (4.2.2.2)

04(C) If the behavior associated with {_POSIX_SAVED_IDS} is supported and the implementation provides the mechanism for creating processes with the appropriate privileges to change user IDs:

When the calling process has the required appropriate privileges to change user IDs, then a call to *setuid*(*uid*) sets the real user ID, effective user ID, and the saved set-user ID to *uid* and returns a value of zero.

05(C) If the behavior associated with {_POSIX_SAVED_IDS} is supported:

When the process does not have the required appropriate privileges to change user IDs, and argument uid is equal to the real user ID or the saved set-user ID, then a call to *setuid*(*uid*) sets the effective user ID to *uid* and returns a value of zero. The real user ID and saved set-user ID remain unchanged.

D01(C) If the implementation provides a method or methods for obtaining the appropriate privileges to change real, effective, and saved setuids:

The method(s) are documented in 2.2.2.4 or in 4.2.2.2 of the PCD.1.

06(C) If the behavior associated with {_POSIX_SAVED_IDS} is not supported and the implementation provides the mechanism for creating processes with the appropriate privileges to change user IDs:

When the calling process has the required appropriate privileges to change user IDs, then a call to *setuid*(*uid*) sets the real user ID and the effective user ID to *uid* and returns a value of zero.

07(C) If the behavior associated with {_POSIX_SAVED_IDS} is not supported:

When the process does not have the required appropriate privileges to change user IDs, and argument *uid* is equal to the real user ID, then a call to *setuid*(*uid*) sets the effective user ID to *uid* and returns a value of zero. The real user ID remains unchanged.

4.2.2.1.3 Returns (4.2.2.3)

R01 When a call to *setuid*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 4–7 in 4.2.2.1.2.)

R02 When a call to *setuid*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error. (See Assertions 8 and 9 in 4.2.2.1.4.)

4.2.2.1.4 Errors (4.2.2.4)

08(C) If the implementation supports an invalid value for uid:

When the value of the *uid* argument is invalid, then a call to *setuid*(*uid*) returns a value of (*int*)-1 and sets *errno* to [EINVAL]. Neither the real nor the effective user ID of the calling process is changed.

09(A) If the behavior associated with {_POSIX_SAVED_IDS} is supported:

When the process does not have the required appropriate privileges to change user IDs, and *uid* does not match either the real user ID or the saved set-user-ID, then a call to *setuid*(*uid*) returns a value of (*int*)-1 and sets *errno* to [EPERM]. Neither the real nor the effective user ID of the calling process is changed.

Otherwise:

When the process does not have the required appropriate privileges to change user IDs, and *uid* does not match the real user ID, then a call to *setuid*(*uid*) returns a value of (*int*)-1 and sets *errno* to [EPERM]. Neither the real nor the effective user ID of the calling process is changed.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.2.2.2 *setgid*() (4.2.2)

4.2.2.2.1 Synopsis (4.2.2.1)

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `int setgid(gid_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *setgid()* is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *setgid*() is defined as a macro when the header <unistd.h> is included:
When the macro *setgid*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *setgid*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.2.2.2.2 Description (4.2.2.2)

04(C) If the behavior associated with {_POSIX_SAVED_IDS} is supported and the implementation provides the mechanism for creating processes with the appropriate privileges to change group IDs:
When the calling process has the required appropriate privileges to change group IDs, then a call to *setgid(gid)* sets the real group ID, effective group ID, and the saved set-group ID to *gid* and returns a value of zero.

05(C) If the behavior associated with {_POSIX_SAVED_IDS} is supported:
When argument *gid* is equal to the real group ID or the saved set-group ID, then a call to *setgid(gid)* sets the effective group ID to *gid* and returns a value of zero. The real group ID, saved set-group ID, and any supplementary group IDs remain unchanged.

D01(C) If the implementation provides a method or methods for obtaining the appropriate privileges to change real and effective group IDs:
The method(s) are documented in 2.2.2.4 or in 4.2.2.2 of the PCD.1.

06(C) If the behavior associated with {_POSIX_SAVED_IDS} is not supported and the implementation provides the mechanism for creating processes with the appropriate privileges to change group IDs:
When a process has the required appropriate privileges to change group IDs then a call to *setgid(gid)* sets the real group ID and the effective group ID to *gid* and returns a value of zero.

07(C) If the behavior associated with (_POSIX_SAVED_IDS) is not supported:
When the process does not have the required appropriate privileges to change group IDs, and argument *gid* is equal to the real group ID, then a call to *setgid(gid)* sets the effective group ID to *gid* and returns a value of zero. The real group ID and any supplementary group IDs remain unchanged.

4.2.2.2.3 Returns (4.2.2.3)

R01 When a call to *setgid*() completes successfully, then a value of (*int*)0 is returned. (see Assertions 4–7 in 4.2.2.2.2.)

R02 When a call to *setgid*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (see Assertions 8 and 9 in 4.2.2.2.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.2.2.2.4 Errors (4.2.2.4)

08(C) If the implementation supports an invalid value for *gid*:

When the value of gid argument is invalid, then a call to *setgid*(*gid*) returns a value of (*int*)-1 and sets *errno* to [EINVAL]. Neither the real nor the effective group ID of the calling process is changed.

09(A) If the behavior associated with {_POSIX_SAVED_IDS} is supported:

When the process does not have the required appropriate privileges to change group IDs, and *gid* does not match either the real group ID or the saved set-group-ID, then a call to *setgid*(*gid*) returns a value of (*int*)-1 and sets *errno* to [EPERM]. Neither the real nor the effective group ID of the calling process is changed.

Otherwise:

When the calling process does not have the required appropriate privileges to change group IDs, and *gid* does not match the real group ID, then a call to *setgid*(*gid*) returns a value of (*int*)-1 and sets *errno* to [EPERM]. Neither the real nor the effective group ID of the calling process is changed.

4.2.3 *getgroups*()

4.2.3.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `int getgroups(int, gid_t *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *getgroups*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *getgroups*() is defined as a macro when the header <unistd.h> is included:

When the macro *getgroups*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *getgroups*() is defined as a macro in the header <unistd.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.2.3.2 Description

04(A) If {NGROUPS_MAX} > 0:

When the value of *gidsetsize* is greater than or equal to the number of supplementary group IDs of the process, then a call to *getgroups*(*gidsetsize*, *grouplist*) places the current list of supplementary group IDs (which may also contain the effective group ID of the calling process) into the array *grouplist* and returns the number of supplementary group IDs placed in *grouplist*.

Otherwise:

A call to *getgroups*(*gidsetsize*, *grouplist*) places no supplementary group IDs into the array *grouplist* and returns a value of zero.

D01(C) If the values of array entries with indices larger than or equal to the returned value from *getgroups*() is documented:

The details are contained in 4.2.3.2 of the PCD.1.

05(A) When *gidsetsize* is zero, then a call to *getgroups*(*gidsetsize*, *grouplist*) returns the number of supplementary group IDs (which may also contain the effective group ID of the calling process) associated with the calling process without modifying the array pointed to by the *grouplist* argument.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

06(A) A call to *getgroups(gidsetsize, grouplist)* returns a value less than or equal to that returned by *sysconf(_SC_NGROUPS_MAX)*.

4.2.3.3 Returns

R01 When a call to *getgroups*() completes successfully, then the number of supplementary group IDs is returned. (See Assertions 4 and 5 in 4.2.3.2.)

R02 When a call to *getgroups*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error. (See Assertion 7 in 4.2.3.4.)

4.2.3.4 Errors

07(C) If a call to *sysconf(_SC_NGROUPS_MAX)* returns a value greater than zero:
When the parameter *gidsetsize* is not equal to zero and is less than the number of supplementary group IDs, then a call to *getgroups(gidsetsize, grouplist)* returns a value of (*int*)-1 and sets *errno* to [EINVAL].

4.2.4 *getlogin*()

4.2.4.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `char * getlogin(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *getlogin*() is declared with the result type char *. (See GA36 in 2.7.3.)

02(C) If *getlogin*() is defined as a macro when the header <unistd.h> is included:
When the macro *getlogin*() is invoked, then it expands to an expression with the result type char *. (See GA37 in 2.7.3.)

03(C) If*getlogin*() is defined as a macro in the header <unistd.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.2.4.2 Description

04(A) A call to*getlogin*() returns a pointer to the login name of the user associated by the login activity with the controlling terminal.

Testing Requirements:
Test for the condition when the real user ID is equal to the effective user ID and when it is not equal to the effective user ID.

4.2.4.3 Returns

R01 When a call to *getlogin*() completes successfully, then a pointer to the login name of the user is returned. (See Assertion 4 in 4.2.4.2.)

05(B) When a call to *getlogin*() completes unsuccessfully because the login name of the user cannot be found, then a **NULL** pointer is returned.

See Reason 2 in Section 5. of POSIX.3 {4}.

D01(C) If the return value from *getlogin*() points to static data that may be overwritten by each *getlogin*() call, and this is documented:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The details are contained in 4.2.4.3 of the PCD.1

4.2.4.4 Errors

D02(C) If it is documented whether error conditions are detected for *getlogin*():
The details are contained in 4.2.4.4 of the PCD.1.

4.3 Process Groups

4.3.1 *getpgrp*()

4.3.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `pid_t getpgrp(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *getpgrp*() is declared with the result type *pid_t*. (See GA36 in 2.7.3.)

02(C) If *getpgrp*() is defined as a macro when the header <unistd.h> is included:
When the macro *getpgrp*() is invoked, then it expands to an expression with the result type *pid_t*. (See GA37 in 2.7.3.)

03(C) If *getpgrp*() is defined as a macro in the header <unistd.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.3.1.2 Description

04(A) A call to *getpgrp*() returns the process group ID of the calling process.

4.3.1.3 Returns

There are no assertions specific to this subclause.

4.3.1.4 Errors

There are no assertions specific to this subclause.

4.3.2 *setsid*()

4.3.2.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `pid_t setsid(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *setsid*() is declared with the result type *pid_t*. (See GA36 in 2.7.3.)

02(C) If *setsid*() is defined as a macro when the header <unistd.h> is included:
When the macro *setsid*() is invoked, then it expands to an expression with the result type *pid_t*. (See GA37 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

03(C) If *setsid*() is defined as a macro in the header <unistd.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.3.2.2 Description

04(A) When the calling process is not a process group leader, then a call to *setsid*() creates a new session, and the value of the process group ID of the calling process is returned.

05(A) The new session created with a call to *setsid*() has the calling process as the session leader. On return from a successful call to *setsid*(), the calling process is the process group leader of a new process group.

06(PCTS_GTI_DEVICE?A:UNTESTED)

When a call to *setsid*() completes successfully, then on return the process has no controlling terminal.

07(A) A call to *setsid*() sets the process group ID of the calling process to the process ID of the calling process.

R01 When a call to *setsid*() completes successfully, then on return the calling process is the only process in the new process group and this calling process is the only process in the new session. (See Assertions 4 and 5 in 4.3.2.2.)

4.3.2.3 Returns

R02 When a call to *setsid*() completes successfully, then the value of the process group ID of the calling process is returned. (See Assertion 4 in 4.3.2.2.)

R03 When a call to *setsid*() completes unsuccessfully, then a value of (*pid_t*)–1 is returned and sets *errno* to indicate the error. (See Assertions 8 and 9 in 4.3.2.4.)

4.3.2.4 Errors

08(PCTS_GTI_DEVICE?A:UNTESTED)

When the calling process is already a process group leader, then a call to *setsid*() returns a value of (*pid_t*)-1 and sets *errno* to [EPERM]. The calling process does not relinquish its controlling terminal.

09(PCTS_GTI_DEVICE?B:UNTESTED)

When the process group ID of a process other than the calling process matches the process ID of the calling process, then a call to *setsid*() returns a value of (*pid_t*)-1 and sets *errno* to [EPERM]. The calling process does not relinquish its controlling terminal.

See Reason 3 in Section 5. of POSIX.3 [4].

4.3.3 *setpgid*()

4.3.3.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `int setpgid(pid_t, pid_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *setpgid*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *setpgid*() is defined as a macro when the header <unistd.h> is included:
When the macro *setpgid*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

03(C) If *setpgid*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.3.3.2 Description

D01(C) If {_POSIX_JOB_CONTROL} is not defined and it is documented whether *setpgid*() is supported:
The documentation is in 4.3.3.2 of the PCD.1.

04(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported:
When *pid* refers to a process that is not a session leader, *pgid* and *pid* are not equal, *pgid* matches the process group ID of a process in the same session as the calling process, *pid* matches the process ID of a process in the same session as the calling process, and *pid* matches the process ID of a child process of the calling process that has not executed an exec function, then a call to *setpgid*() sets the process group ID of the indicated process to *pgid* and returns a value of zero.

05(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported:
When *pid* refers to a process that is not a session leader, *pgid* and *pid* are equal, *pid* matches the process ID of a process in the same session as the calling process, and *pid* matches the process ID of a child process of the calling process that has not executed an exec function, then a call to *setpgid*() sets the process group ID of the indicated process to *pgid* and returns a value of zero.

06(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported:
When *pid* refers to a process that is not a session leader, *pgid* matches the process group ID of a process in the same session as the calling process, and *pid* is equal to zero, then a call to *setpgid*() sets the process group ID of the calling process to *pgid* and returns a value of zero.

07(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported:
When *pid* refers to a process that is not a session leader, *pid* matches the process ID of a process in the same session as the calling process, *pgid* is equal to zero, and *pid* matches the process ID of a child process of the calling process that has not executed an exec function, then a call to *setpgid*() sets the process group ID of the target process to *pid* and returns a value of zero.

08(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported:
When the *pid* of the calling process is not a session leader, and when both the *pid* and *pgid* are set to zero, then a call to *setpgid*() sets the group ID of the calling process equal to its process ID and returns a value of zero.

R01 A call to *setpgid*() returns a value of (*int*)−1 and sets *errno* to [ENOSYS]. (See Assertion 12 in 4.3.3.4.)

4.3.3.3 Returns

R02 When a call to *setpgid*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 4–8 in 4.3.3.2.)

R03 When a call to *setpgid*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error. (See Assertions 9–16 in 4.3.3.4.)

4.3.3.4 Errors

09(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported:
When the value of the *pid* argument matches the process ID of a child process of the calling process, and when the child process has successfully executed an exec() function, then a call to set *pgid*(*pid*, *pgid*) returns a value of (*int*)-1 and sets *errno* to [EACCES]. The process group ID of the calling process is not changed.

10(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the value of the *pgid* argument is less than zero, then a call to set *pgid(pid, pgid)* returns a value of (*int*)-1 and sets *errno* to [EINVAL]. The process group ID of the calling process is not changed.

11(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported and the implementation supports an invalid value when *pgid* is greater than 0:
When the value of the *pgid* argument is invalid, then a call to set *pgid(pid, pgid)* returns a value of (*int*)-1 and sets *errno* to [EINVAL]. The process group ID of the calling process is not changed.

12(C) If the behavior associated with {_POSIX_JOB_CONTROL} is not supported and *setpgid*() is not supported:
A call to *setpgid*() returns a value of (*int*)-1 and sets *errno* to [ENOSYS]. The process group ID of the calling process is not changed.

13(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported:
When the value of the *pid* argument matches the process ID of a session leader, then a call to set*pgid(pid, pgid)* returns a value of (*int*)-1 and sets *errno* to [EPERM]. The process group ID of the calling process is not changed.

14(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported:
When the value of the *pid* argument matches the process ID of a child process of the calling process, and when the child process is not in the same session as the calling process, then a call to *setpgid(pid, pgid)* returns a value of (*int*)-1 and sets *errno* to [EPERM]. The process group ID of the calling process is not changed.

15(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported:
When the value of the *pgid* argument does not match the process ID of the process indicated by the pid argument, and when there is no process with a process group ID that matches the value of pgid argument in the same session as the calling process, then a call to *setpgid(pid, pgid)* returns a value of (*int*)-1 and sets *errno* to [EPERM]. The process group ID of the calling process is not changed.

16(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *setpgid*() is supported:
When the value of the pid argument does not match the process ID of the calling process or a child process of the calling process, then a call to *setpgid(pid, pgid)* returns a value of (*int*)-1 and sets *errno* to [ESRCH]. The process group ID of the calling process is not changed.

4.4 System Identification

4.4.1 *uname*()

4.4.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <sys/utsname.h> is included, then the function prototype `int uname(struct utsname *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <sys/utsname.h> is included, then the function *uname*() is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *uname*() is defined as a macro when the header <sys/utsname.h> is included:
When the macro *uname*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *uname*() is defined as a macro in the header <sys/utshame.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.4.1.2 Description

04(A) A call to *uname*(name) stores information identifying the current operating system in a structure pointed to by name and returns a nonnegative value.

05(A) When the header <sys/utsname.h> is included, then the data items *sysname*, *nodename*, *release*, *version*, and *machine* are members of *struct utsname*.

06(A) When a call to *uname*() is successful, then the *sysname*, *nodename*, *release*, *version*, and *machine* elements of the *utsname* structure referenced by name are each null-terminated character arrays.

D01(A) The format of each member of *struct utsname* defined in Table 4-1 of POSIX.1 {3} is described in 4.4.1.2 of the PCD.1.

4.4.1.3 Returns

R01 When a call to *uname*() completes successfully, then a nonnegative value is returned. (See Assertion 4 in 4.4.1.2.)

07(B) When a call to *uname*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error.

See Reason 2 in Section 5. of POSIX.3 {4}.

4.4.1.4 Errors

D02(C) If error conditions are detected for *uname*(), and this is documented:
The details are contained in 4.4.1.4 of the PCD.1.

4.5 Time

4.5.1 *time*()

4.5.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <time.h> is included, then the function prototype `time_time(time_t *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <time.h> is included, then the function *time*() is declared with the result type *time_t*. (See GA36 in 2.7.3.)

02(C) If *time*() is defined as a macro when the header <time.h> is included:
When the macro *time*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *time_t*. (See GA37 in 2.7.3.)

03(C) If *time*() is defined as a macro in the header <time.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.5.1.2 Description

04(A) When *tloc* is NULL or when *tloc* is a valid address, then a call to *time*(*tloc*) returns the value of time in seconds since the Epoch.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

05(A) When *tloc* is other than a **NULL** pointer, then a call to *time(tloc)* causes the return value to be stored in the location pointed to by *tloc*.

4.5.1.3 Returns

R01 When a call to *time*() completes successfully, then the value of time is returned. (See Assertions 4 and 5 in 4.5.1.2.)

06(B) When a call to *time*() completes unsuccessfully, then a value of (*time_t*)-1 is returned and sets *errno* to indicate the error.

See Reason 2 in Section 5. of POSIX.3 {4}.

4.5.1.4 Errors

D01(c) If the implementation documents whether and under what conditions errors are detected for *time*():
The details are contained in 4.5.1.4 of the PCD.1.

4.5.2 *times*()

4.5.2.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <sys/times.h> is included, then the function prototype `clock_t times(struct tms *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <sys/times.h> is included, then the function *times*() is declared with the result type *clock_t*. (See GA36 in 2.7.3.)

02(C) If *times*() is defined as a macro when the header <sys/times.h> is included:
When the macro *times*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *clock_t*. (See GA37 in 2.7.3.)

03(C) If *times*() is defined as a macro in the header <sys/times.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.5.2.2 Description

04(A) When the header <sys/times.h> is included, then the *structure tms* is defined and contains at least the members *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* of type *clock_t*.

R01 All members of the structure pointed to by buffer, after a call to *times(buffer)*, contains information reported in units of clock ticks. (See Assertions 6–9 in 4.5.2.2.)

05(A) When a call to *times*() completes successfully, then the elapsed real time in clock ticks from an arbitrary point in the past is returned.

R02 A call to *times(buffer)* from a parent process fills the structure pointed to by buffer with accounting information that contains the *tms_cutime* and *tms_cstime* times for processes for which *wait*() or *waitpid*() have been called, and that does not contain *tms_cutime* and *tms_cstime* times for those processes for which the call to *wait*() or *waitpid*() has not waited. (See Assertions 8 and 9 in 4.5.2.2.)

06(A) A call to *times(buffer)* fills the structure pointed to by buffer with accounting information, and the value *tms_utime* in clock ticks is the CPU time charged for the execution of user instructions while executing instructions of the calling process.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

07(A) A call to *times(buffer)* fills the structure pointed to by buffer with accounting information, and the value *tms_stime* in clock ticks is the CPU time charged for the execution by the system on behalf of the calling process.

08(A) A call to *times(buffer)* fills the structure pointed to by buffer with accounting information that contains the *tms_cutime*, in clock ticks, which is the sum of the *tms_utime* and *tms_cutime* of all terminated child processes for which *wait*() or *waitpid*() has been called, and that does not contain the *tms_utime* and *tms_cutime* times for those processes for which the call to *wait*() or *waitpid*() has not waited.

09(A) A call to *times(buffer)* fills the structure pointed to by buffer with accounting information that contains the *tms_cstime* in clock ticks, which is the sum of the *tms_stime* and *tms_cstime* of all terminated child processes for which *wait*() or *waitpid*() has been called, and that does not contain the *tms_stime* and *tms_cstime* times for those processes for which the call to *wait*() or *waitpid*() has not waited.

4.5.2.3 Returns

R03 When a call to *times*() completes successfully, then the elapsed real time in clock ticks from an arbitrary point in the past is returned. (See Assertion 5 in 4.5.2.2.)

10(A) A call to *times*() keeps the arbitrary point in the past as a constant between invocations of *times*() within the same process.

11(B) When a call to *times*() completes unsuccessfully, then a value of (*clock_t*)-1 is returned and sets *errno* to indicate the error.

See Reason 2 in Section 5. of POSIX.3 {4}.

D01(C) If it is documented whether the return value from *times*() may overflow the range of type *clock_t*:
The details are contained in 4.5.2.3 of the PCD.1.

4.5.2.4 Errors

D02(C) If the implementation documents whether and under what conditions errors are detected for *times*():
The details are contained in 4.5.2.4 of the PCD.1.

4.6 Environment Variables

4.6.1 *getenv*()

4.6.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <stdlib.h> is included, then the function prototype
`char * getenv(const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdlib.h> is included, then the function *getenv*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

02(C) If *getenv*() is defined as a macro when the header <stdlib.h> is included:
When the macro *getenv*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

03(C) If *getenv*() is defined as a macro in the header <stdlib.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.6.1.2 Description

04(A) A call to *getenv(name)* returns a pointer to the value associated with the environment variable name.

05(A) A call to *getenv(name)* returns a **NULL** pointer for names that do not exist in the environment.

06(A) Upper- and lowercase letters in the environment retain their unique identities.

07(A) Environment variable names support the portable filename character set.

4.6.1.3 Returns

R01 When a call to *getenv*() completes successfully, then either a pointer to a string containing the environment variable is returned or, if the environment variable requested cannot be found, a **NULL** pointer is returned. (See Assertions 4 and 5 in 4.6.1.2.)

08(D) If the implementation supports error conditions for *getenv*():
When a call to *getenv*() completes unsuccessfully, then a **NULL** pointer is returned to indicate the error.

See Reason 2 in Section 5. of POSIX.3 {4}.

D01(C) If it is documented whether the return value from *getenv*() points to static data that may be overwritten by subsequent calls to *getenv*():
The details are contained in 4.6.1.3 of the PCD.1.

4.6.1.4 Errors

D02(C) If the implementation documents whether and under what conditions errors are detected for *getenv*():
The details are contained in 4.6.1.4 of the PCD.1.

4.7 Terminal Identification

4.7.1 *ctermid*()

4.7.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `char * ctermid(char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *ctermid*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

02(C) If *ctermid*() is defined as a macro when the header <stdio.h> is included:
When the macro *ctermid*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

03(C) If *ctermid*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.7.1.2 Description

04(PCTS_GTI_DEVICE?A:UNTESTED)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A call to *ctermid*() generates a string that, when used. as a pathname, refers to the current controlling terminal for the current process.

05(PCTS_GTI_DEVICE?A:UNTESTED)

When a call to *ctermid((char *) 0)* completes successfully, then a pointer to a string is returned that, when used as a pathname, refers to the current controlling terminal of the process.

4.7.1.3 Returns

06(A) When the header <stdio.h> is included, then the symbolic constant L_ctermid is defined and has a value greater than zero.

07(PCTS_GTI_DEVICE?A:UNTESTED)

When a call to *ctermid*(s) completes successfully and s is not a **NULL** pointer, then a string that, when used as a pathname, refers to the current controlling terminal is placed in the character array to which s points, the length of the string pointed to by s is less than L_ctermid bytes, and the value of s is returned.

08(B) When a call to *ctermid*() completes unsuccessfully, then an empty string is returned.

See Reason 2 in Section 5. of POSIX.3 {4}.

D01(C) If it is documented whether the argument to *ctermid*() is a **NULL** pointer when the return value from *ctermid*() points to static data that may be overwritten by subsequent calls to *ctermid()*:
The details are contained in 4.7.1.3 of the PCD.1.

4.7.1.4 Errors

D02(C) If the implementation documents whether and under what conditions errors are detected for *ctermid*():
The details are contained in 4.7.1.4 of the PCD.1.

4.7.2 Determinable Terminal Device Name

4.7.2.1 *ttyname*() (4.7.2)

4.7.2.1.1 Synopsis (4.7.2.1)

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `char * ttyname(int)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *ttyname*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

02(C) If *ttyname*() is defined as a macro when the header <unistd.h> is included:
When the macro *ttyname*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

03(C) If *ttyname*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.7.2.1.2 Description (4.7.2.2)

04(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *ttyname(fildes)* returns a pointer to a string containing a null-terminated pathname for the terminal associated with file descriptor *fildes*.

D01(C) If it is documented whether the return value from *ttyname*() points to static data that may be overwritten by subsequent calls to *ttyname*():
The details are contained in 4.7.2.2 of the PCD.1.

4.7.2.1.3 Returns (4.7.2.3)

05(A) When the file descriptor argument is not a valid file descriptor associated with a terminal device, then a call to *ttyname(fildes)* completes unsuccessfully, and a **NULL** pointer is returned.

06(B) When the pathname cannot be determined, then a call to *ttyname(fildes)* completes unsuccessfully, and a **NULL** pointer is returned.

See Reason 3 in Section 5. of POSIX.3 [4].

4.7.2.1.4 Errors (4.7.2.4)

D02(C) If the implementation documents whether and under what conditions errors are detected for *ttyname*():
The details are contained in 4.7.2.4 of the PCD.1.

4.7.2.2 *isatty*() (4.7.2)

4.7.2.2.1 Synopsis (4.7.2.1)

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `int isatty(int)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *isatty*() is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *isatty*() is defined as a macro when the header <unistd.h> is included:
When the macro *isatty*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *isatty*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.7.2.2.2 Description (4.7.2.2)

04(PCTS_GTI_DEVICE?A:UNTESTED)

When the argument *fildes* is a valid file descriptor associated with a terminal, then a call to *isatty(fildes)* returns a vale of 1.

05(A) When the argument *fildes* is not a file descriptor associated with a terminal, then a call to *isatty(fildes)* returns a value of 0.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.7.2.2.3 Returns (4.7.2.3)

There are no assertions specific to this subclause.

4.7.2.2.4 Errors (4.7.2.4)

D01(C) If the implementation documents whether and under what conditions errors are detected for *isatty*():
The details are contained in 4.7.2.4 of the PCD.1.

4.8 Configurable System Variables

4.8.1 *sysconf*() (4.8.1)

4.8.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `long sysconf(int)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *sysconf*() is declared with the result type *long*. (See GA36 in 2.7.3.)

02(C) If *sysconf*() is defined as a macro when the header <unistd.h> is included:
When the macro *sysconf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *long*. (See GA37 in 2.7.3.)

03(C) If *sysconf*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

4.8.1.2 Description

04(A) When the header <unistd.h> is included, then the symbolic constants _SC_ARG_MAX, _SC_CHILD_MAX, _SC_CLK_TCK, _SC_NGROUPS_MAX, _SC_OPEN_MAX, _SC_JOB_CONTROL, _SC_SAVED_IDS, _SC_VERSION, _SC_STREAM_MAX, and _SC_TZNAME_MAX are defined and are distinct.

05(A) If ARG_MAX is defined when the header <limits.h> is included:
A call to *sysconf*(*_SC_ARG_MAX*) either returns -1 without changing *errno* or returns a value greater than or equal to {ARG_MAX}.

Otherwise:
A call to *sysconf*(*_SC_ARG_MAX*) either returns -1 without changing *errno* or returns a value greater than or equal to {_POSIX_ARG_MAX}.

06(A) If CHILD_MAX is defined when the header <limits.h> is included:
A call to *sysconf*(*_SC_CHILD_MAX*) either returns -1 without changing *errno* or returns a value greater than or equal to {CHILD_MAX}.

Otherwise:
A call to *sysconf*(*_SC_CHILD_MAX*) either returns -1 without changing *errno* or returns a value greater than or equal to {_POSIX_CHILD_MAX}.

07(A) A call to *sysconf*(*_SC_CLK_TCK*) returns a value equal to the value returned by the macro CLK_TCK, defined in <time.h>.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

08(A) A call to *sysconf(_SC_NGROUPS_MAX)* returns a value greater than or equal to the value of NGROUPS_MAX in <limits.h>.

09(A) If OPEN_MAX is defined when the header <limits.h> is included:
A call to *sysconf(_SC_OPEN_MAX)* either returns -1 without changing *errno* or returns a value greater than or equal to {OPEN_MAX}.

Otherwise:
A call to *sysconf(_SC_OPEN_MAX)* either returns -1 without changing *errno* or returns a value greater than or equal to {_POSIX_OPEN_MAX}.

10(A) If STREAM_MAX is defined when the header <limits.h> is included:
A call to *sysconf(_SC_STREAM_MAX)* either returns -1 without changing *errno* or returns a value greater than or equal to {STREAM_MAX}.

Otherwise:
A call to *sysconf(_SC_STREAM_MAX)* either returns -1 without changing *errno* or returns a value greater than or equal to {_POSIX_STREAM_MAX}.

11(A) If TZNAME_MAX is defined when the header <limits.h> is included:
A call to *sysconf(_SC_TZNAME_MAX)* either returns -1 without changing *errno* or returns a value greater than or equal to {TZNAME_MAX}.

Otherwise:
A call to *sysconf(_SC_TZNAME_MAX)* either returns -1 without changing *errno* or returns a value greater than or equal to {_POSIX_TZNAME_MAX}.

12(A) If {_POSIX_JOB_CONTROL} is defined when the header <unistd.h> is included:
A call to *sysconf(_SC_JOB_CONTROL)* returns a value other than -1.

Otherwise:
A call to *sysconf(_SC_JOB_CONTROL)* does not change the value of *errno*.

13(A) If {_POSIX_SAVED_IDS} is defined when the header <unistd.h> is included:
A call to *sysconf(_SC_SAVED_IDS)* returns a value other than -1.

Otherwise:
A call to *sysconf(_SC_SAVED_IDS)* does not change the value of *errno*.

14(A) A call to *sysconf(_SC_VERSION)* returns a value equal to {_POSIX_VERSION}.

D01(C) If the implementation supports variables beyond those listed in Table 4-2 of POSIX.1 {3}, and this is documented:
The details are contained in 4.8.1.2 of the PCD.1.

4.8.1.3 Returns

R01 When a call to *sysconf(name)* completes unsuccessfully because argument name is an invalid value, then a value of (*long*)-1 is returned. (See Assertion 18 in 4.8.1.4.)

15(C) If the behavior associated with {_POSIX_JOB_CONTROL} is not supported:
A call to *sysconf(_SC_JOB_CONTROL)* returns a value of (*long*)-1, and the value of *errno* is not changed.

16(C) If the behavior associated with {_POSIX_SAVED_IDS} is not supported:
A call to *sysconf(_SC_SAVED_IDS)* returns a value of (*long*)-1, and the value of *errno* is not changed.

R02 When a call to *sysconf(name)* completes successfully and the variable corresponding to the argument *name* is defined on the system, then the current variable value on the system is returned. (See Assertions 5–14 in 4.8.1.2.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17(B) When a call to *sysconf(name)* completes successfully, then the value returned does not change during the lifetime of the calling process.

See Reason 3 in Section 5. of POSIX.3 {4}.

4.8.1.4 Errors

18(A) When the value of the argument *name* is invalid, then a call to *sysconf(name)* returns a value of(*long*)-1 and sets *errno* to [EINVAL].

4.8.2 Special Symbol {CLK_TCK} (4.8.1.5)

R03 The special symbol CLK_TCK yields the same result as a call to *sysconf(_SC_CLK_TCK)*. (See Assertion 7 in 4.8.1.2.)

19(A) When the header <time.h> is included, then the special symbol CLK_TCK is defined.

5. Files and Directories

5.1 Directories

5.1.1 Format of Directory Entries

01(A) When the header <dirent.h> is included, then the structure *dirent* is defined.

D01(C) If the internal format of directories is documented:
The details are contained in 5.1.1 of the PCD.1.

02(A) The *struct dirent* includes the member *d_name* whose type is *char*[].

D02(C) If the size of the array *d_name* is documented:
The details are contained in 5.1.1 of the PCD.1.

03(B) When a call to *readdir*() completes successfully, then the number of bytes preceding the null character in the array *d_name* does not exceed {NAME_MAX} bytes.

See Reason 1 in Section 5. of POSIX.3 {4}.

5.1.2 Directory Operations

5.1.2.1 *opendir*() (5.1.2)

5.1.2.1.1 Synopsis (5.1.2.1)

01(A) If the implementation provides C Standard {2} support:
When the header <dirent.h> is included, then the function prototype `DIR * opendir(const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <dirent.h> is included, then the function *opendir*() is declared with the result type *DIR* *. (See GA36 in 2.7.3.)

02(C) If *opendir*() is defined as a macro when the header <dirent.h> is included:
When the macro *opendir*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *DIR* *. (See GA37 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

03(C) If *opendir*() is defined as a macro in the header <dirent.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.1.2.1.2 Description (5.1.2.2)

04(A) When the header <dirent.h> is included, then the type *DIR* is defined.

R01 When an *exec* function is called, then the directory stream is closed in the new process image. (See Assertion 38 in 3.1.2.2.)

05(A) When the first filename component of the *dirname* argument is ".", and the pathname does not begin with a slash, then *opendir*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

06(A) When the *dirname* argument points to the string "/", then *opendir*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

07(A) When the *dirname* argument points to the string "///", then *opendir*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

08(A) When the *dirname* argument points to a string beginning with a single slash or beginning with three or more slashes, then *opendir*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (see GA17 in 2.3.6.)

09(A) When the first filename component of the *dirname* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *opendir*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

10(A) When the *dirname* argument points to the string "F1/" and F1 is a directory, then *opendir*() resolves the pathname to F1, which is in the current working directory. (See GA19 in 2.3.6.)

11(A) When the argument *dirname* points to the string "F1//" and F1 is a directory, then *opendir*() resolves the pathname to F1, which is in the current working directory. (See GA20 in 2.3.6.)

12(A) When the *dirname* argument points to the string "F1/F2", then *opendir*() resolves the pathname to the file F2 in the directory F1, which is in the current working directory. (See GA21 in 2.3.6.)

13(A) When the pathname argument points to the string " F1/./F2", then *opendir*() resolves the pathname to the file F2 in the directory F1, which is in the current working directory. (See GA22 in 2.3.6.)

14(A) When the *dirname* argument points to the string "F1/../F1/F2", then *opendir*() resolves the pathname to the file F2 in the directory F1, which is in the current working directory. (see GA23 in 2.3.6.)

15(A) When the *dirname* argument points to the string "F1//F2", then *opendir*() resolves the pathname to the file F2 in the directory F1, which is in the current working directory. (See GA24 in 2.3.6.)

16(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:
When the *dirname* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *opendir*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

17(A) On a call to *opendir*(*dirname*), the pathname *dirname* supports filenames containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18(A) On a call to *opendir(dirname)*, the pathname *dirname* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

19(A) When *opendir(dirname)* is granted read access to *dirname* and search access to each component of the path prefix of *dirname*, then the standard file access control mechanism does not cause *opendir*() to indicate that file access is denied.

20(A) A call to *opendir(dirname)* opens a directory stream corresponding to the directory named by the *dirname* argument.

21(A) A call to *opendir*() positions the directory stream at the first directory entry.

D01(C) If the action taken when a file is removed from or added to the directory after the most recent call to *opendir*() is documented:

The details on whether a subsequent call to *readdir*() returns an entry for that file is contained in 5.1.2.2 of the PCD.1.

5.1.2.1.3 Returns (5.1.2.3)

R02 When a call to *opendir*() completes successfully, then an object of type *DIR* * is returned. (See Assertions 5–19 in 5.1.2.1.2.)

R03 When a call to *opendir*() completes unsuccessfully, then a value of NULL is returned and sets *errno* to indicate the error. (See Assertions 22–24, 26–29, 31, and 33 in 5.1.2.1.4.)

5.1.2.1.4 Errors (5.1.2.4)

22(A) When search permission is denied for any component of the path prefix of *dirname*, then a call to *opendir (dirname)* returns a value of NULL and sets *errno* to [EACCES].

23(A) When read permission is denied for *dirname*, then a call to *opendir(dirname)* returns a value of NULL and sets *errno* to [EACCES].

24({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *dirname* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *opendir(dirname)* returns a value of NULL and sets *errno* to [ENAMETOOLONG].

25({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *dirname* equals {PCTS_NAME_MAX}, then a call to *opendir(dirname)* succeeds.

26(A) If {PATH_MAX}≤{PCTS_PATH_MAX}:

When the length of the *dirname* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *opendir(dirname)* returns a value of NULL and sets *errno* to [ENAMETOOLONG].

Otherwise:

When the length of the path argument is {PCTS_PATH_MAX}, then a call to *opendir*() is successful.

27(A) When the argument *dirname* is not an existing directory, then a call to *opendir(dirname)* returns a value of NULL and sets *errno* to [ENOENT].

28(A) When the argument *dirname* points to an empty string, then a call to *opendir(dirname)* returns a value of NULL and sets *errno* to [ENOENT].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

29(A) When a component of the directory argument *dirname* is not a directory, then a call to *opendir(dirname)* returns a value of NULL and sets *errno* to [ENOTDIR].

D02(C) If the implementation supports the detection of [EMFILE] for *opendir*():
The details under which [EMFILE] occurs for *opendir*() are contained in 5.1.2.4 of the PCD.1. (See DGA02 in 2.4.)

30({OPEN_MAX}>{PCTS_OPEN_MAX}?A:UNTESTED)

{PCTS_OPEN_MAX} files or directories can be opened.

31({OPEN_MAX}≤{PCTS_OPEN_MAX}?C:UNTESTED)

If {PCD_TYPE_DIR} is **TRUE** and the implementation supports the detection of [EMFILE] for *opendir*():
When {OPEN_MAX} file descriptors have been opened, then a subsequent call to *opendir (dirname)* returns a value of NULL and sets *errno* to [EMFILE].

32({OPEN_MAX}≤{PCTS_OPEN_MAX}?D:UNTESTED)

If {PCD_TYPE_DIR} is not documented and the implementation supports the detection of [EMFILE] for *opendir*():
When {OPEN_MAX} file descriptors have been opened, then a subsequent call to *opendir (dirname)* will either succeed, returning a value of non-NULL, or will fail, returning a value of NULL and setting *errno* to [EMFILE].

See Reason 2 in Section 5. of POSIX.3 {4}.

33({OPEN_MAX}≤{PCTS_OPEN_MAX}?D:UNTESTED)

If {PCD_TYPE_DIR} is **TRUE** and the implementation does not support the detection of [EMFILE] for *opendir*():
When [OPEN_MAX] files have been opened, then a subsequent call to *opendir(dirname)* is successful (unless a different error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 {4}.

D03(C) If the implementation supports the detection of [ENFILE] for *opendir*():
The details under which [ENFILE] occurs for *opendir*() are contained in 5.1.2.4 of the PCD.1. (See DGA02 in 2.4.)

34(system limit on open files is ≤ {PCTS_OPEN_MAX}?D:UNTESTED)

If {PCD_TYPE_DIR} is **TRUE** and the implementation supports the detection of [ENFILE] for *opendir*():
When the limit on the number of open files on the system has been opened, then a subsequent call to *opendir (dirname)* returns a value of NULL and sets *errno* to [ENFILE].

See Reason 4 in Section 5. of POSIX.3 {4}.

35(system limit on open files is ≤ {PCTS_OPEN_MAX}? D:UNTESTED)

If {PCD_TYPE_DIR} is **TRUE** and the implementation does not support the detection of [ENFILE] for *opendir*():
When the limit on the number of open files on the system has been opened, then a subsequent call to *opendir (dirname)* is successful (unless a different error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 {4}.

5.1.2.2 *readdir*() (5.1.2)

5.1.2.2.1 Synopsis (5.1.2.1)

01(A) If the implementation provides C Standard {2} support:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the header <dirent.h> is included, then the function prototype
`struct dirent * readdir(DIR *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <dirent.h> is included, then the function *readdir*() is declared with the result type *struct dirent* *. (See GA36 in 2.7.3.)

02(C) If *readdir*() is defined as a macro when the header <dirent.h> is included:
When the macro *readdir*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *struct dirent* *. (See GA37 in 2.7.3.)

03(C) If *readdir*() is defined as a macro in the header <dirent.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.1.2.2.2 Description (5.1.2.2)

04(A) A call to *readdir(dirp)* returns a pointer to a structure representing the entry at the current position in the directory stream.

Testing Requirements:
Test for file names consisting of 1 to {NAME_MAX} bytes in length.

05(A) A call to *readdir*() advances the directory stream to the next directory entry.

06(A) A call to *readdir*() upon reaching the end of the directory stream returns a **NULL** pointer, and *errno* is not changed.

07(B) A call to *readdir*() does not return directory entries containing empty names.

See Reason 3 in Section 5. of POSIX.3 {4}.

D01(C) If it is documented whether the pointer returned by *readdir* () points to static data that may be overwritten by subsequent calls to *readdir*() on the same directory stream:
The details are contained in 5.1.2.2 of the PCD.1.

08(A) The structure representing the directory entry associated with one directory stream is not overwritten by a call to *readdir*() on a different directory stream.

D02(C) If it is documented whether the *readdir*() function buffers several directory entries per actual read operation:
The details are contained in 5.1.2.2 of the PCD.1.

09(A) When a call to *readdir*() is made that causes a read to the underlying directory, then the *st_atime* time-related field of the directory is marked for update.

10(A) When a call to *fork*() is successful and the child process does not continue to process the directory stream, then the parent may continue processing the directory stream using *readdir*() and *rewinddir*() or both.

11(A) When a call to *fork*() is successful and the parent process does not continue to process the directory stream, then the child may continue processing the directory stream using *readdir*() and *rewinddir*() or both.

D03(C) If the action taken when using a directory stream after one of the `exec` type function calls is documented:
The details on its effect are contained in 5.1.2.2 of the PCD.1.

D04(C) If the action taken when both the parent and child processes call *readdir*() after a call to *fork*() is documented:
The details on its effect are contained in 5.1.2.2 of the PCD.1.

5.1.2.2.3 Returns (5.1.2.3)

R01 When a call to *readdir*() completes successfully, then it returns an object of type *struct dirent* *. (See Assertion 1 in 5.1.2.2.1.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

R02 When a call to *readdir*() completes successfully and the end of directory is encountered, then a **NULL** pointer is returned and *errno* is not changed. (See Assertion 6 in 5.1.2.2.1.)

R03 When a call to *readdir*() completes unsuccessfully, then a value of NULL is returned and sets *errno* to indicate the error. (See Assertion 12 in 5.1.2.2.4.)

5.1.2.2.4 Errors (5.1.2.4)

D05(C) If the action associated with passing a *dirp* argument to *readdir*() that does not refer to a currently open directory stream is documented:
The details on its effect are contained in 5.1.2.2 of the PCD.1.

D06(C) If the implementation supports the detection of [EBADF] for *readdir*():
The details under which [EBADF] occurs for *readdir*() are contained in 5.1.2.4 of the PCD.1. (See DGA02 in 2.4.)

12(C) If {PCD_DIR_TYPE} is **TRUE** and the implementation supports the detection of [EBADF] *for readdir*():
When the *dirp* argument does not refer to an open directory stream, then a call to *readdir*(*dirp*) returns a value of NULL and sets *errno* to [EBADF].

13(D) If the implementation does not support the detection of [EBADF] for *readdir*():
A call to *readdir*(*dirp*) is successful (unless a different error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 {4}.

5.1.2.3 *rewinddir*() (5.1.2)

5.1.2.3.1 Synopsis (5.1.2.1)

01(C) If the implementation provides C Standard {2} support:
When the header <dirent.h> is included, then the function prototype `void rewinddir(DIR *)` is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *rewinddir*() is contained in 5.1.2.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(D) If the implementation provides C Standard {2} support and *rewinddir*() is defined as a macro when the header <dirent.h> is included:
When the macro *rewinddir*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)

See Reason 2 in Section 5. of POSIX.3 {4}.

03(C) If *rewinddir*() is defined as a macro in the header <dirent.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 in 1.3.3.)

5.1.2.3.2 Description (5.1.2.2)

04(A) A call to *rewinddir*(*dirp*) resets the position of the named directory stream to which *dirp* refers to the beginning of the directory.

05(A) A call to *rewinddir*(*dirp*) causes the directory stream to refer to the current state of the corresponding directory, as a call to *opendir*() would have done.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D02(C) If the action taken when a file is removed from or added to the directory after the most recent call to *rewinddir*() is documented:

The details on whether a subsequent call to *readdir*() returns an entry for that file is contained in 5.1.2.2 of the PCD.1.

D03(C) If the action associated with passing a *dirp* argument to *rewinddir*() that does not refer to a currently open directory stream is documented:

The details on its effect are contained in 5.1.2.2 of the PCD.1.

D04(C) If the action taken when parent and child processes both continue processing of a directory stream with a combination of *readdir*() and *rewinddir*() is documented:

The details on its effect are contained in 5.1.2.2 of the PCD.1.

5.1.2.3.3 Returns (5.1.2.3)

There are no assertions specific to this subclause.

5.1.2.3.4 Errors (5.1.2.4)

There are no assertions specific to this subclause.

5.1.2.4 *closedir*() (5.1.2)

5.1.2.4.1 Synopsis (5.1.2.1)

01(A) If the implementation provides C Standard {2} support:

When the header <dirent.h> is included, then the function prototype `int closedir(DIR *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <dirent.h> is included, then the function *closedir*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *closedir*() is defined as a macro when the header <dirent.h> is included:

When the macro *closedir*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *closedir*() is defined as a macro in the header <dirent.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.1.2.4.2 Description (5.1.2.2)

04(A) When *closedir*() is called on an open directory stream, then it returns a value of zero and makes the directory stream no longer accessible to *readdir*().

05(C) If {PCD_DIR_TYPE} is **TRUE**:

A call to *closedir*() closes the referenced file descriptor.

D01(C) If whether a directory pointer *dirp* continues to point to an accessible object of type DIR after return from a call to *closedir*(*dirp*) is documented:

The details are contained in 5.1.2.2 of the PCD.1.

D02(C) If the action associated with passing a *dirp* argument to *closedir*() that does not refer to a currently open directory stream is documented:

The details on its effect are contained in 5.1.2.2 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5.1.2.4.3 Returns (5.1.2.3)

R01 When a call to *closedir*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 4 in 5.1.2.4.2.)

R02 When a call to *closedir*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertion 6 in 5.1.2.4.4.)

5.1.2.4.4 Errors (5.1.2.4)

D03(C) If the implementation supports the detection of [EBADF] for *closedir*():
The details under which [EBADF] occurs for *closedir*() are contained in 5.1.2.4 of the PCD.1. (See DGA02 in 2.4.)

06(C) If {PCD_DIR_TYPE} is **TRUE** and the implementation supports the detection of [EBADF] for *closedir*():
When the *dirp* argument does not refer to an open directory stream, then a call to *closedir*(*dirp*) returns a value of (*int*)-1 and sets *errno* to [EBADF].

07(D) If the implementation does not support the detection of [EBADF] for *closedir*():
A call to *closedir*(*dirp*) is successful (unless a different error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 {4}.

5.2 Working Directory

5.2.1 *chdir*()

5.2.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `int chdir(const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *chdir*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *chdir*() is defined as a macro when the header <unistd.h> is included:
When the macro *chdir*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *chdir*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.2.1.2 Description

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *chdir*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(A) When the *path* argument points to the string "/", then *chdir*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

06(A) When the *path* argument points to the string "///", then *chdir*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *chdir*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *chdir*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *path* argument points to the string "F1/" and `F1` is a directory, then *chdir*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

10(A) When the argument *path* points to the string "F1//" and `F1` is a directory, then *chdir*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *path* argument points to the string "F1/F2", then *chdir*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the pathname argument points to the string "F1/./F2", then *chdir*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string "F1/../F1/F2", then *chdir*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string "F1//F2", then *chdir*() resolves the pathname to the file `F2` in the directory `F1`. which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:
When the *path* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *chdir*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *chdir(path)*, the pathname *path* supports *filenames* containing any of the following characters:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *chdir(path)*, the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) When *chdir(path)* is granted search access to all components of *path*, then the standard file access control mechanism does not cause *chdir*() to indicate that file access is denied.

19(A) A call to *chdir(path)* causes the named directory argument path to become the current working directory and returns a zero value.

5.2.1.3 Returns

R01 When a call to *chdir*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 19 in 5.2.1.2.)

R02 When a call to *chdir*() completes unsuccessfully, then a value of (*int*)-1 is returned, sets *errno* to indicate the error, and the current working directory remains unchanged. (See Assertions 20, 21, and 23–27 in 5.2.1.4.)

5.2.1.4 Errors

20(A) When search permission is denied for a component of the path, then a call to *chdir(path)* returns a value of (*int*)-1 and sets *errno* to [EACCES] The current directory remains unchanged.

21({NAME_MAX}≤{PCTS_NAME_MAX}C:UNTESTED)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:
When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename ({NAME_MAX}), then a call to *chdir(path)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG]. The current directory remains unchanged.

22({NAME_MAX}>{PCTS_NAME_MAX}C:UNTESTED)

If the behavior associated with {POSIX_NO_TRUNC} is supported for the file:
When the length of a pathname component of *path* equals {PCTS_NAME_MAX}, then a call to *chdir(path)* succeeds.

23(A) If {PATH_MAX} ≤ {PCTS_PATH MAX}:
When the length of the *path* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *chdir(path)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG]. The current directory remains unchanged.

Otherwise:
When the length of the *path* argument is {PCTS_PATH_MAX}, then a call to *chdir()* is successful.

24(A) When a component of path is not a directory, then a call to *chdir(path)* returns a value of (*int*)-1 and sets *errno* to [ENOTDIR]. The current directory remains unchanged.

25(A) When path does not exist, then a call to *chdir(path)* returns a value of (*int*)-1 and sets *errno* to [ENOENT]. The current directory remains unchanged.

26(A) When a component of the path prefix of path does not exist, then a call to *chdir(path)* returns a value of (*int*)-1 and sets *errno* to [ENOENT]. The current directory remains unchanged.

27(A) When path is an empty string, then a call to *chdir(path)* returns a value of (*int*)-1 and sets *errno* to [ENOENT]. The current directory remains unchanged.

5.2.2 *getcwd*()

5.2.2.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype
`char * getcwd(char *, size_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *getcwd*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

02(C) If *getcwd*() is defined as a macro when the header <unistd.h> is included:
When the macro *getcwd*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

03(C) If *getcwd*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.2.2.2 Description

04(A) A call to *getcwd(buf, size)* copies the absolute pathname of the current working directory into the character array *buf* and returns a pointer to be result.

D01(C) If the behavior of *getcwd*() when *buf* is a **NULL** pointer is documented:
The details are contained in 5.2.2.2 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.2.2.3 Returns

R01 When a call to *getcwd*() completes successfully, then the *buf* argument is returned. (See Assertion 4 in 5.2.2.2.)

R02 When a call to *getcwd*() completes unsuccessfully, then a **NULL** pointer is returned and sets *errno* to indicate the error. (See Assertions 5–8 in 5.2.2.4.)

D02(C) If the contents of the buffer passed to *getcwd*() after an error is documented:
The details are contained in 5.2.2.3 of the PCD.1.

5.2.2.4 Errors

05(A) When the size argument is equal to zero, then a call to *getcwd(buf, size)* returns a value of NULL and sets *errno* to [EINVAL].

06(A) When the size argument is greater than zero but smaller than the length of the pathname plus 1, then a call to *getcwd(buf, size)* returns a value of NULL and sets *errno* to [ERANGE].

D03(C) If the implementation supports the detection of [EACCES] for *getcwd*():
The details under which [EACCES] occurs for *getcwd*() are contained in 5.2.2.4 of the PCD.1. (See DGA02 in 2.4.)

07(A) When read permission is denied for a component of the pathname, then a call to *getcwd(buf, size)* either returns a value of NULL and sets *errno* to [EACCES] or returns the current working directory into the character array *buf*.

08(A) When search permission is denied for a component of the pathname, then a call to *getcwd(buf, size)* either returns a value of NULL and sets *errno* to [EACCES] or returns the current working directory into the character array *buf*.

5.3 General File Creation

5.3.1 *open*()

5.3.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <fcntl.h> is included, then the function prototype
`int open(const char *, int, ...)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <fcntl.h> is included, then the function *open*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *open*() is defined as a macro when the header <fcntl.h> is included:
When the macro *open*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *open*() is defined as a macro in the header <fcntl.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5.3.1.2 Description

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *open*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(A) When the *path* argument points to the string "/", then *open*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

06(A) When the *path* argument points to the string "///", then *open*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *open*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6)

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *open*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *path* argument points to the string "F1/" and `F1` is a directory, then *open*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

10(A) When the argument *path* points to the string "F1//" and `F1` is a directory, then open() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *path* argument points to the string "F1/F2", then *open*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the pathname argument points to the string "F1/./F2", then *open*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string "F1/../F1/F2", then *open*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string "F1//F2", then *open*() resolves the pathname to the file `F2` in the directory `F1` which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

When the *path* component is a string of more than {NAME MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *open*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *open*(*path, oflag*), the pathname *path* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *open*(*path, oflag*), the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) When *open*(*path, O_RDONLY*) is granted read access to *path* and search access to the path prefix of *path*, then the standard file access control mechanism does not cause *open*() to indicate that file access is denied.

D01(C) If the result of a call to *open*() on a FIFO with O_RDWR set is documented:

The details are contained in 5.3.1.2 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

19(A) When *open(path, O_WRONLY)* is granted write access to *path* and search access to the path prefix of *path*, then the standard file access control mechanism does not cause *open*() to indicate that file access is denied.

20(A) When *open(path, O_WRONLY | O_CREAT | O_EXCL, mode)* is granted write and search access to the path prefix of *path*, then the standard file access control mechanism does not cause *open*() to indicate that file access is denied.

21(A) A call to *open*() returns a file descriptor that can be used by other I/O functions to refer to the file specified by argument *path*.

22(A) A call to *open*() returns a file descriptor for the named file that is the lowest file descriptor not currently open for that process.

23(B) The file descriptor returned by *open*() refers to an open file description that is not shared with any other process in the system.

See Reason 3 in Section 5. of POSIX.3 [4].

24(A) The file offset associated with the file descriptor returned by *open*() is initially set to the beginning of the file.

25(A) A call to *open*() returns a file descriptor that has the FD_CLOEXEC flag clear.

26(A) A file descriptor returned by a call to *open(path, O_RDONLY)* can be read and cannot be written.

27(A) A file descriptor returned by a call to *open(path, O_WRONLY)* can be written and cannot be read.

28(A) A file descriptor returned by a call to *open(path, O_RDWR)* can be read and written.

R01 A file descriptor returned by a call to *open(path, O_WRONLY|O_APPEND)* or *open(path, O_RDWR|O_APPEND)* has the file offset set to end of file prior to each write. (See Assertion 10 in 6.4.2.)

29(A) When the O_TRUNC flag is not set and the file exists, then a call to *open*() does not change the permission bits, ownership, data of the file, *st_atime*, *st_ctime*, or *st_mtime* fields.

30(A) When the argument *path* does not reference an existing file and the O_CREAT flag is set, then a call to *open*() creates the file.

31(A) The user ID of a newly created file is set to the effective user ID of the process.

D02(C) If the conditions under which the group ID of the new file is set to the group ID of the directory in which the file is being created and the conditions under which the group ID of the file is set to the effective group ID of the calling process for a successful call to *open*() with O_CREAT set are documented:
The details are contained in 5.3.1.2 of the PCD.1.

32(A) The group ID of a newly created file is set to either the effective group ID of the process, or the group ID of the directory in which the file is created.

33(A) When the O_CREAT flag is set and the file does not exist, then a call to *open*() sets the file permission bits of the newly created file to the value of mode except for those bits set in the file mode creation mask of the process.

D03(C) If the effect when *open*() is called with O_CREAT set and when bits in the third argument other than the mode bits are set is documented:
The details are contained in 5.3.1.2 of the PCD.1.

R02 When the O_EXCL and O_CREAT flags are set and the file already exists, then a call to *open*() fails. (See Assertion 52 in 5.3.1.4.)

34(B) When more than one process is executing a call to *open*() for the same file with the O_CREAT and O_EXCL flags set, then the check for the existence of the file and the creation of the file is atomic.

See Reason 3 in Section 5. of POSIX.3 [4].

D04(C) If the action taken when *open*() is called with O_EXCL set and O_CREAT not set is documented:
The details are contained in 5.3.1.2 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

R03 When the O_NOCTTY flag is set and *path* identifies a terminal device, then a call to *open*() does not cause the terminal device to become the controlling terminal for the process. (See Assertion 4 in 7.1.1.3.)

35(A) When the O_RDONLY and O_NONBLOCK flags are set and the named file is a FIFO, then a call to *open*() returns without delay.

R04 When the O_WRONLY and O_NONBLOCK flags are set, the named file is a FIFO, and no process has the file opened for reading, then a call to *open*() returns an error. (See Assertion 64 in 5.3.1.4.)

36(A) When the O_RDONLY flag is set, the O_NONBLOCK flag is clear, and the named file is a FIFO, then a call to *open*() blocks until a process opens the file for writing.

37(A) When the O_WRONLY flag is set, the O_NONBLOCK flag is clear, and the named file is a FIFO, then a call to *open*() blocks until a process opens the file for reading.

38(D) If the implementation supports block special files with nonblocking I/O:

When the O_NONBLOCK flag is set and the named file is such a block special file, then a call to *open*() returns without waiting for availability of the device.

See Reason 1 in Section 5. of POSIX.3 [4].

39(D) If the implementation supports character special files with nonblocking I/O:

When the O_NONBLOCK flag is set and the named file is such a character special file, then a call to *open*() returns without waiting for availability of the device.

NOTE — The case of a terminal device file is covered by Assertion 57 in 7.1.2.4.

See Reason 1 in Section 5. of POSIX.3 [4].

D05(C) If a block special or character special device that supports nonblocking opens is called by *open*() with O_NONBLOCK set, and the behavior of the device after the *open*() call is documented:

The details are contained in 5.3.1.2 of the PCD.1.

40(D) If the implementation supports block special files with nonblocking I/O:

When the O_NONBLOCK flag is clear and the named file is such a block special file, then a call to *open*() blocks until the device is available.

See Reason 1 in Section 5. of POSIX.3 [4].

41(D) If the implementation supports character special files with nonblocking I/O:

When the O_NONBLOCK flag is clear and the named file is such a character special file, then a call to *open*() blocks until the device is available.

NOTE — The case of a terminal device file is covered by Assertion 56 in 7.1.2.4.

See Reason 1 in Section 5. of POSIX.3 [4].

D06(C) If the result is documented of a call to *open(path, oflag,...)* when path is not a block or character file that supports nonblocking *open*()s, and when argument *path* is not a FIFO being opened with O_RDONLY or O_WRONLY set in *oflag*:

The details are contained in 5.3.1.2 of the PCD.1.

42(A) When an existing regular file is successfully opened with flags O_RDWR | O_TRUNC or with flags O_WRONLY | O_TRUNC set, then the file is truncated to zero length, and the mode and owner are unchanged.

43(A) When a call to *open(path, oflag)* is made with the O_TRUNC flag set in *oflag*, and when *path* refers to a FIFO special file, then the result of the call to *open*() is the same as with the O_TRUNC flag clear.

44(PCTS_GTI_DEVICE?A:UNTESTED)

When a call to *open(path, oflag)* is made with the O_TRUNC flag set in *oflag*, and when *path* refers to a terminal device file, then the result of the call to *open*() is the same as with the O_TRUNC flag clear.

Testing Requirements:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Test on the input buffer and, when the general terminal interface device of the implementation buffers output, also test on the output buffer.

D07(A) The details of the effect of O_TRUNC on file types other than regular files, FIFO special files, or terminal device files is contained in 5.3.1.2 of the PCD.1.

D08(C) If the effect of calling *open*() with O_TRUNC and O_RDONLY set is documented:
The details are contained in 5.3.1.2 of the PCD.1.

R05 The file descriptor returned by a successful call to *open*() is set to remain open across `exec` function calls. (See Assertion 25 in 5.3.1.2.)

45(A) When O_CREAT is set and the file did not previously exist, then a successful call to *open*() marks for update the *st_atime*, *st_ctime*, and *st_mtime* fields on the created file, and the *st_ctime* and *st_mtime* fields on the parent directory.

46(A) When O_TRUNC is set and the file named by *path* is an existing regular file, then a successful call to *open*() marks for update the *st_ctime* and *st_mtime* fields on the file.

5.3.1.3 Returns

R06 When a call to *open*() completes successfully, then a nonnegative integer that represents the lowest numbered unused file descriptor is returned. (See Assertion 22 in 5.3.1.2.)

R07 When a call to *open*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error, and no files are created or modified. (See Assertions 47–56 and 58–67 in 5.3.1.4.)

5.3.1.4 Errors

47(A) When search permission is denied for a component of the path prefix of *path*, then a call to *open*(*path*, *oflag*) returns a value of (*int*)-1 and sets *errno* to [EACCES].

48(A) When the file exists and the requested read and/or write permission is denied, then a call to *open*(*path*, *oflag*) returns a value of (*int*)-1 and sets *errno* to [EACCES].

49(A) When the O_CREAT flag is specified, the file does not exist, and write permission is denied for the parent directory of the file to be created, then a call to *open*(*path*, *oflag*) returns a value of (*int*)-1, sets *errno* to [EACCES], and the file is not created.

50(A) When the O_TRUNC flag is specified and write permission is denied for the file, then a call to *open*(*path*, *oflag*) returns a value of (*int*)-1, sets *errno* to [EACCES], and the file is not truncated.

51(B) When O_NONBLOCK is set, the named file is not a FIFO or terminal device file, and the program would be delayed in the open operation, then a call to *open*() returns a value of (*int*)-1 and sets *errno* to [EAGAIN].

See Reason 1 in Section 5. of POSIX.3 {4}.

52(A) When *oflag* argument has O_CREAT and O_EXCL set and the file named by *path* exists, then a call to *open* (*path*, *oflag*) returns a value of (*int*)-1, sets *errno* to [EEXIST], and does not mark for update either the *st_ctime* and *st_mtime* fields of the file and parent directory or the *st_atime* field of the file.

53(A) When the open operation is interrupted by a signal, then a call to *open*(*path*, *oflag*) returns a value of (*int*)-1, sets *errno* to [EINTR], and does not mark for update either the *st_ctime* and *st_mtime* fields of the file and parent directory or the *st_atime* field of the file.

54(A) When the file named by *path* is a directory and *oflag* is O_WRONLY or O_RDWR, then a call to *open*(*path*, *oflag*) returns a value of (*int*)-1, sets *errno* to [EISDIR], and does not mark for update either the *st_ctime* and *st_mtime* fields of the file and parent directory or the *st_atime* field of the file.

55(A) If {OPEN_MAX} ≤ {PCTS_OPEN_MAX}:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When {OPEN_MAX} files have been opened, then a subsequent call to *open(path, oflag, mode)* returns a value of (*int*)-1 and sets *errno* to [EMFILE].

Testing Requirements:

Test for instances

1) When the file does not exist and the O_CREAT flag specifies that the file is not created, and
2) When the file exists and the O_TRUNC flag specifies that the file is not truncated

Otherwise:

{PCTS_OPEN_MAX} files can be opened.

56({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *open(path, oflag)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

57({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED

If the behavior associated with {_POSIX_NO TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS_NAME_MAX}, then a call to *open(path, oflag)* succeeds.

58(A) If {PATH_MAX}≤{PCTS_PATH_MAX}:

When the length of the *path* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *open(path, oflag)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

Otherwise:

When the length of the path argument is {PCTS_PATH_MAX}, then a call to *open*() is successful.

59(B) When too many files are currently open in the system, then a call to *open(path, oflag)* returns a value of (*int*)-1 and sets *errno* to [ENFILE].

Testing Requirements:

Test for instances

1) When the file does not exist and the O_CREAT flag specifies that the file is not created, and
2) When the file exists and the O_TRUNC flag specifies that the file is not truncated

See Reason 1 in Section 5. of POSIX.3 {4}.

60(A) When the *oflag* argument does not have O_CREAT set and the file named by *path* does not exist, then a call to *open(path, oflag)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

61(A) When the path prefix of *path* does not exist or *path* points to an empty string, then a call to *open(path, oflag)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

62(B) When the directory or file system that would contain the new file cannot be extended, then a call to *open(path, oflag)* returns a value of (*int*)-1, sets *errno* to [ENOSPC], and the file is not created.

See Reason 1 in Section 5. of POSIX.3 {4}.

63(A) When a component of the path prefix of *path* is not a directory, then a call to *open(path, oflag)* returns a value of (*int*)-1 and sets *errno* to [ENOTDIR].

64(A) When O_WRONLY and O_NONBLOCK are set, the file named by *path* is a FIFO, and no process has the file opened for reading, then a call to *open(path, oflag)* returns a value of (*int*)-1 and sets *errno* to [ENXIO].

65(C) If the implementation supports a read-only file system:

When the named file resides on a read-only file system and *oflag* has O_WRONLY or O_RDWR, then a call to *open(path, oflag)* returns a value of(*int*)-1 and sets *errno* to [EROFS].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

66(C) If the implementation supports a read-only file system:
When the named file is to reside on a read-only file system, when *oflag* has O_WRONLY or O_RDWR in combination with O_CREAT set, and when the file does not exist, then a call to *open*(*path*, *oflag*) returns a value of (*int*)-1, sets *errno* to [EROFS], does not mark for update the *st_ctime* and *st_mtime* fields of the parent directory, and the file is not created.

67(C) If the implementation supports a read-only file system:
When the named file resides on a read-only file system, when *oflag* has O_TRUNC set, and when the file does exist, then a call to *open*(*path*, *oflag*) returns a value of (*int*)-1, sets *errno* to [EROFS], does not truncate the file, and does not mark for update the *st_atime*, *st_ctime*, and *st_retiree* fields of the file. (See GA13 in 2.3.5.)

5.3.2 *creat*()

5.3.2.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <fcntl.h> is included, then the function prototype
`int creat(const char *, mode_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <fcntl.h> is included, then the function *creat*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *creat*() is defined as a macro when the header <fcntl.h> is included:
When the macro *creat*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *creat*() is de fined as a macro in the header <fcntl.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.3.2.2 Description

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *creat*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(B) When the *path* argument points to the string "/", then *creat*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

06(B) When the *path* argument points to the string "///", then *creat*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *creat*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *creat*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

09(B) When the *path* argument points to the string "F1/" and `F1` is a directory, then *creat* () resolves the pathname to F1, which is in the current working directory. (See GA19 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 [4].

10(B) When the *path* argument points to the string "F1//" and `F1`, is a directory, then *creat*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 [4].

11(A) When the *path* argument points to the string "F1/F2", then *creat*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the *path* argument points to the string "F1/./F2", then *creat*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string "F1/../F1/F2", then *creat*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string "F1//F2", then *creat*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:
When the *path* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *creat*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *creat*(*path*, *mode*), the pathname *path* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9. _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *creat*(*path*, *mode*), the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) When *creat*(*path*, ...) is granted write access to the existing file *path* and search access to the path prefix of *path*, then the standard file access control mechanism does not cause *creat*() to indicate that file access is denied.

19(A) When *creat*(*path*, ...) is granted write access to the parent directory of the nonexistent file *path* and search access to the path prefix of *path*, then the standard file access control mechanism does not cause *creat*() to indicate that file access is denied.

20(A) A call to *creat*() returns a file descriptor that can be used by other I/O functions to refer to the file specified by argument *path*.

21(A) A call to *creat*() returns a file descriptor for the named file that is the lowest file descriptor not currently open for that process.

22(B) The file descriptor returned by *creat*() is not shared with any other process in the system.

See Reason 3 in Section 5. of POSIX.3 [4].

23(A) The file offset associated with the file descriptor returned by *creat*() is initially set to the beginning of the file.

24(A) A call to *creat*() returns a file descriptor that has the FD_CLOEXEC flag clear.

25(A) A file descriptor opened by *creat*() can be written and cannot be read.

26(A) When the file exists, then a call to *creat*() does not change any of the following: permission bits, ownership, and *st_atime*.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

27(A) When argument path does not reference an existing file, then a call to *creat*() creates the file.

28(A) The user ID of a newly created file is set to the effective user ID of the process.

29(A) The group ID of a newly created file is set to either the effective group ID of the process or the group ID of the directory in which the file is created.

30(A) When the file does not exist, then a call to *creat*() sets the file permission bits of a newly created file to the value of mode except for those set in the file mode creation mask of the process.

31(A) When the named file is a FIFO, then a call to *creat*() blocks until a process opens the file for reading.

32(D) If the implementation supports block special files with nonblocking I/O:
When the named file is a block special file, then a call to *creat*() blocks until the device is available.

See Reason 1 in Section 5. of POSIX.3 {4}.

33(D) If the implementation supports character special files with nonblocking I/O:
When the named file is a character special file, then a call to *creat*() blocks until the device is available.
NOTE — The case of a terminal device file is covered by Assertion 56 in 7.1.2.4.

See Reason 1 in Section 5. of POSIX.3 {4}.

34(A) When an existing regular file is successfully created, then the file is truncated to zero length, and the mode and owner are unchanged.

35(A) When the *path* argument on a call to *creat*(*path*) refers to a FIFO special file, then the result of the call to *creat*() is the same as for the equivalent call to *open*() with the O_TRUNC flag clear or set.

36(PCTS_GTI_DEVICE?A:UNTESTED)

When the *path* argument on a call to *creat*(*path*) refers to a terminal device file, then the result of the call to *creat*() is the same as for the equivalent call to *open*() with the O_TRUNC flag clear or set.

Testing Requirements:
Test on the input buffer and when the general terminal interface device of the implementation buffers output, also test on the output buffer.

R01 The created file descriptor is set to remain open across `exec` function calls. (See Assertion 24 in 5.3.2.2.)

37(A) When the file did not previously exist, then a successful call to *creat*() marks for update the *st_atime*, *st_ctime*, and *st_mtime* fields on the created file, and the *st_ctime* and *st_mtime* fields on the parent directory.

38(A) When the file already exists, then a successful call to *creat*() marks for update the *st_ctime* and the *st_mtime* fields on the file.

5.3.2.3 Returns

R02 When a call to *creat*() completes successfully, then a nonnegative integer that represents the lowest numbered unused file descriptor is returned. (See Assertion 21 in 5.3.2.2.)

R03 When a call to *creat*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error, and no files are created or are created or modified. (See Assertions 39–45 and 47–53 in 5.3.2.4.)

5.3.2.4 Errors

39(A) When search permission is denied for a component of the path prefix of *path*, then a call to *creat*(*path*, *mode*) returns a value of (*int*)-1 and sets *errno* to [EACCES].

40(A) When the file exists and write permission is denied on the file, then a call to *creat*(*path*, *mode*) returns a value of (*int*)-1, sets *errno* to [EACCES], and does not truncate the file.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

41(A) When the file does not exist and write permission is denied for the parent directory of the file to be created, then a call to *creat(path, mode)* returns a value of (*int*)-1, sets *errno* to [EACCES], and does not create the file.

42(A) When the file creation is interrupted by a signal, then a call to *creat(path, mode)* returns a value of (*int*)-1, sets *errno*, to [EINTR], and does not mark for update either the *st_ctime* and *st_mtime* fields of the file and parent directory or the *st_atime* field of the file.

43(A) When the file named by path is a directory, then a call to *creat(path, mode)* returns a value of (*int*)-1, sets *errno* to [EISDIR], and does not mark for update either the *st_ctime* and *st_mtime* fields of the file and parent directory or the *st_atime* field of the file.

44(A) If {OPEN_MAX}≤{PCTS_OPEN_MAX}:

When {OPEN_MAX} files have been opened, then a subsequent call to *creat(path, mode)* returns a value of (*int*)-1 and sets *errno* to [EMFILE].

Testing Requirements:

Test for instances

1) When the file does not exist, it is not created, and
2) When the file exists, it is not truncated

Otherwise:

{PCTS_OPEN_MAX} files can be opened.

45({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *creat(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

46({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS_NAME_MAX}, then a call to *creat(path, mode)* succeeds.

47(A) If {PATH_MAX} ≤ {PCTS_PATH_MAX}:

When the length of the *path* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *creat(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

Otherwise:

When the length of the *path* argument is {PCTS_PATH_MAX}, then a call to *creat*() is successful.

48(B) When too many files are currently open in the system, then a call to *creat(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENFILE].

Testing Requirements:

Test for instances

1) When the file does not exist, it is not created, and
2) When the file exists, it is not truncated

See Reason 1 in Section 5. of POSIX.3 [4].

49(A) When the path prefix of *path* does not exist or *path* points to an empty string, then a call to *creat(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

50(B) When the directory or file system that would contain the new file cannot be extended, then a call to *creat(path, mode)* returns a value of (*int*)-1, sets *errno* to [ENOSPC], and does not create the file.

See Reason 1 in Section 5. of POSIX.3 [4].

51(A) When a component of the path prefix of *path* is not a directory, then a call to *creat(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOTDIR].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860261 228 ■

52(C) If the implementation supports a read-only file system:
When the file exists and the file named by *path* resides on a read-only file system, then a call to *creat*(*path*, *mode*) returns a value of (*int*)-1, sets *errno* to [EROFS], does not truncate the file, and does not mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the file. (See GA13 in 5.2.3.)

53(C) If the implementation supports a read-only file system:
When the named file is to reside on a read-only file system and the file does not exist, then a call to *creat*(*path*, *mode*) returns a value of (*int*)-1, sets *errno* to [EROFS], does not mark for update the *st_ctime* and *st_mtime* fields of the parent directory, and does not create the file. (See GA13 in 2.3.5.)

5.3.3 *umask*()

5.3.3.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <sys/stat.h> is included, then the function prototype `mode_t umask(mode_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <sys/stat.h> is included, then the function *umask*() is declared with the result type *mode_t*. (See GA36 in 2.7.3.)

02(C) If *umask*() is defined as a macro when the header <sys/stat.h> is included:
When the macro *umask*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *mode_t*. (See GA37 in 2.7.3.)

03(C) If *umask*() is defined as a macro in the header <sys/stat.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.3.3.2 Description

04(A) A call to *umask*(*cmask*) sets the file mode creation mask argument of the process to cmask and returns the previous value of the file mode creation mask.

D01(A) The meaning of bits other than file permission bits in the argument to *umask*() is stated in 5.3.3.2 of the PCD.1.

05(A) A call to *umask*(*cmask*) sets the file mode creation mask argument to cmask. Bits set in the file mode creation mask of the process are used to clear the corresponding permission bits in the mode argument supplied by *open*(*path*, *flags*, *mode*), *creat*(*path*, *mode*), *mkdir*(*path*, *mode*), and *mkfifo*(*path*, *mode*).

5.3.3.3 Returns

R01 A call to *umask*() returns the previous value of the file mode creation mask. (See Assertion 4 in 5.3.3.2.)

06(A) When *cmask* is the return value from a previous call to *umask*(), then a call to *umask*(*cmask*) restores the file mode creation mask of the process to the same value as that prior to the previous call to *umask*().

5.3.3.4 Errors

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.3.4 *link*()

5.3.4.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype
`int link(const char *, const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *link*() is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *link*() is defined as a macro when the header <unistd.h> is included:
When the macro *link*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *link*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.3.4.2 Description

04(A) When the first filename component of the existing argument is ".", and the pathname does not begin with a slash, then *link*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05({PCD_LINK_TO_DIRECTORY} is **TRUE**?B:UNTESTED)

When the existing argument points to the string "/", then *link* () resolves the pathname to the root directory of the process. (see GA15 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

06({PCD_LINK_TO_DIRECTORY} is **TRUE**?B:UNTESTED)

When the existing argument points to the string "/ / /", then *link*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

07(A) When the *existing* argument points to a string beginning with a single slash or beginning with three or more slashes, then *link*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *existing* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *link*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(PCTS_APP_LINK_DIR?C:UNTESTED)

If {PCD_LINK_TO_DIRECTORY} is **TRUE**:
When the *existing* argument points to the string "F1/" and `F1` is a directory, then *link*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

10(PCTS_APP_LINK_DIR?C:UNTESTED)

If {PCD_LINK_TO DIRECTORY} is **TRUE**:
When the *existing* argument points to the string "F1//" and `F1` is a directory, then *link*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11(A) When the *existing* argument points to the string "F1/F2", then *link*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the *existing* argument points to the string " F1/./F2", then *link*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *existing* argument points to the string " F1/../F1/F2", then *link*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *existing* argument points to the string " F1//F2", then *link*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

When the *existing* component is a string of more than {_NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *link*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *link*(*existing, new*), the pathname *existing* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *link*(*existing, new*), the pathname *existing* retains the unique identity of its upper- and lowercase letters (See GA03 in 2.2.2.32.)

18(A) When the first filename component of the *new* argument is ".", and the pathname does not begin with a slash, then *link*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

19(B) When the *new* argument points to the string "/", then *link*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

20(B) When the *new* argument points to the string "///", then *link*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

21(A) When the *new* argument points to a string beginning with a single slash or beginning with three or more slashes, then *link*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

22(A) When the first filename component of the *new* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *link*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

23(C) If {PCD_LINK TO_DIRECTORY} is **TRUE**:

When the *new* argument points to the string "F1/"and `F1` is a directory, then *link*() resolves the pathname to F1, which is in the current working directory. (See GA19 in 2.3.6.)

24(C) If {PCD_LINK_TO DIRECTORY} is **TRUE**:

When the *new* argument points to the string "F1//" and `F1` is a directory, then *link*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

25(A) When the *new* argument points to the string "F1/F2", then *link*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

26(A) When the *new* argument points to the string " F1/./F2", then *link*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

27(A) When the *new* argument points to the string "F1/../F1/F2", then *link*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

28(A) When the *new* argument points to the string "F1//F2", then *link*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

29(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:
When the *new* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *link*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

30(A) On a call to *link*(*existing, new*), the pathname *new* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

31(A) On a call to *link*(*existing, new*), the pathname *new* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

32(A) When *link*(*existing, new*) is granted search access to the path prefix of both *existing* and *new* and write permission to the directory that will contain *new*, then the standard file access control mechanism does not cause *link*() to indicate that file access is denied.

33(A) A call to *link*() creates a new link for the existing file such that the new link to the file allows access via *new* to the same data accessible via existing. The call to *link*() returns a value of zero.

34(A) A successful call to *link*(*existing, new*) increments by one the link count of the file associated with *existing*.

35(B) When a call to *link*(*existing, new*) creates a new link, then the creation of the link and the increment of the link count is atomic.

See Reason 2 in Section 5. of POSIX.3 [4].

D01(C) If support or nonsupport of *link*() across file systems is documented:
The details are contained in 5.3.4.2 of the PCD.1.

36(A) After a call to *link*(*existing, new*), the new link, *new*, refers to the same file as the link *existing*.

R01 An unsuccessful call to *link*(*existing, new*) creates no link, and the link count of the file referenced by *existing* remains unchanged. (See Assertion 42 in 5.3.4.4.)

D02(C) If the implementation supports a method to associate the appropriate privileges for linking to directories with a process:
The details are documented in 2.2.2.4 or in 5.3.4.2 of the PCD.1.

D03(C) If support or nonsupport of *link*()on directories is documented:
The details are contained in 5.3.4.2 of the PCD.1.

37(C) If {PCD_LINK TO_DIRECTORY} is **TRUE**:
When the calling process has the appropriate privileges to link to a directory, then a call. to *link*() creates a link to a directory.

38(C) If {PCD_LINK_TO_DIRECTORY} is not documented:
When the file named by *existing* is a directory and the calling process has the appropriate privileges to link to a directory, then a call to *link*(*existing, new*) either creates a link to a directory or returns a value of (*int*)-1, sets *errno* to [EPERM], creates no link, and does not change the link count of the file referenced by *existing*.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D04(C) If support or nonsupport for the condition when the calling process requires permission to access the existing file for *link*() to succeed is documented:
The details are contained in 5.3.4.2 of the PCD.1.

R02 When the calling process does not have permission to access the existing file, then a call to *link*(*existing, new*) returns a value of (*int*)–1 and sets *errno* to [EACCES]. (See Assertions 41–44 in 5.3.4.4.)

39(A) A call to *link*() marks for update the *st_ctime* field of the file and the *st_ctime* and *st_retiree* field of the parent directory of the newly created link.

40(C) If {PCD_LINK_FILE_SYSTEM} is **TRUE**:
When the link named by *new* and the file named by *existing* are on different file systems, then a call to *link*(*existing, new*) is successful.

R03 When the link named by *new* and the file named by *existing* are on different file systems, then a call to *link*(*existing, new*) is either successful or returns a value of (*int*)-1, sets *errno* to [EXDEV], creates no link, and does not change the link count of the file referenced by *existing*. (See Assertions 65 and 66 in 5.3.4.4.)

5.3.4.3 Returns

R04 When a call to *link*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 33 in 5.3.4.2.)

R05 When a call to *link*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertions 41–47, 49, and 51–66 in 5.3.4.4.)

5.3.4.4 Errors

41(A) When search permission is denied for a component of the path prefix of *existing*, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [EACCES], and creates no link.

42(A) When search permission is denied for a component of the path prefix of *new*, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [EACCES], creates no link, and does not change the link count of the file referenced by *existing*.

43(A) When the requested link requires writing in a directory with a mode that denies write permission, then a call to *link* (*existing, new*) returns a value of (*int*)-1, sets *errno* to [EACCES], and does not change the link count of the file referenced by *existing*.

44(D) If the implementation requires access permission to link to the file:
When the calling process does not have permission to access the existing file, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [EACCES], and creates no link.

See Reason 2 in Section 5. of POSIX.3 {4}.

45(A) When the link named by *new* exists, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [EEXIST], and does not change the link count of the file referenced by *existing*.

46(A) If {LINK_MAX} ≤ {PCTS_LINK_MAX}:
When {LINK_MAX} links to the file named by the argument *existing* already exist, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [EMLINK], creates no link, and does not change the link count of the file referenced by *existing*.

Otherwise:
{PCTS_LINK_MAX} links can be created to the file specified by argument *existing*.

47({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO TRUNC} is supported for the file:
When the length of a pathname component of *existing* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *link* (*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENAMETOOLONG], and creates no link.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

48({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:
When the length of a pathname component of *existing* equals {PCTS_NAME_MAX}, then a call to *link*(*existing, new*) succeeds.

49({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:
When the length of a pathname component of *new* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENAMETOOLONG], and does not change the link count of the file referenced by *existing*.

50({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:
When the length of a pathname component of *new* equals {PCTS_NAME_MAX}, then a call to *link*(*existing, new*) succeeds.

51(A) If {PATH_MAX} ≤ {PCTS_PATH_MAX}:
When the length of the *existing* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENAMETOOLONG], and creates no link.

Otherwise:
When the length of the path argument is {PCTS_PATH_MAX}, then a call to *link*() is successful.

52(A) If {PATH_MAX} ≤ {PCTS PATH_MAX}:
When the length of the new argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENAMETOOLONG], and does not change the link count of the file referenced by *existing*.

Otherwise:
When the length of the path argument is {PCTS_PATH_MAX}, then a call to *link*() is successful.

53(A) When a component of the path prefix of *existing* does not exist, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENOENT], and creates no link.

54(A) When a component of the path prefix of *new* does not exist, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENOENT], and does not change the link count of the file referenced by existing.

55(A) When the file named by *existing* does not exist, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENOENT], and creates no link.

56(A) When *existing* points to an empty string, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENOENT], and creates no link.

57(A) When *new* points to an empty string, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENOENT], and does not change the link count of the file referenced by *existing*.

58(B) When the directory that would contain the link cannot be extended, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENOSPC], creates no link, and does not change the link count of the file referenced by *existing*.

See Reason 1 in Section 5. of POSIX.3 {4}.

59(A) When a component of the path prefix of *existing* is not a directory, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENOTDIR], and creates no link.

60(A) When a component of the path prefix of *new* is not a directory, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [ENOTDIR], and does not change the link count of the file referenced by *existing*.

61(C) If {PCD_LINK_TO_DIRECTORY} is **TRUE**:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When the file named by *existing* is a directory and the calling process does not have the appropriate privileges to link a directory, then a call to *link* (*existing, new*) returns a value of (*int*)-1, sets *errno* to [EPERM], creates no link, and does not change the link count referenced by *existing*.

62(C) If {PCD_LINK_TO_DIRECTORY} is **FALSE**:
When the file named by *existing* is a directory, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [EPERM], creates no link, and does not change the link count of the file referenced by *existing*.

63(C) If {PCD_LINK_TO_DIRECTORY} is not documented:
When the file named by *existing* is a directory and the calling process does not have the appropriate privileges to link to a directory, then a call to *link* (*existing, new*) returns a value of (*int*)-1, sets *errno* to [EPERM], creates no link, and does not change the link count of the file referenced by *existing*.

64(C) If the implementation supports a read-only file system:
When the requested link requires writing in a directory on a read-only file system, then a call to *link* (*existing, new*) returns a value of (*int*)-1, sets *errno* to [EROFS], creates no link, does not change the link count of the file referenced by *existing*, and does not mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the parent directory of new. (See GA13 in 5.2.3.)

65(C) If {PCD_LINK_FILE_SYSTEM} is **FALSE**:
When the link named by *new* and the file named by existing are on different file systems, then a call to *link*(*existing, new*) returns a value of (*int*)-1, sets *errno* to [EXDEV], creates no link, and does not change the link count, of the file referenced by existing.

66(C) If {PCD_LINK_FILE_SYSTEM} is not documented:
When the link named by *new* and the file named by existing are on different file systems:, then a call to *link*(*existing, new*) is eithert successful or returns a value of (*int*)-1, sets *errno* to [EXDEV], creates no link, and does not change the link count of the file referenced by existing.

5.4 Special File Creation

5.4.1 *mkdir*()

5.4.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <sys/stat.h> is included, then the function prototype
`int mkdir(const char *, mode_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <sys/stat.h> is included, then the function *mkdir*() is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *mkdir*() is defined as a macro when the header <sys/stat.h> is included:
When the macro *mkdir*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type int. (See GA37 in 2.7.3.)

03(C) If *mkdir*() is defined as a macro in the header <sys/stat.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5.4.1.2 Description

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *mkdir*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(B) When the *path* argument points to the string " /", then *mkdir*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

06(B) When the *path* argument points to the string "/ / /", then *mkdir*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *mkdir*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *mkdir*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(B) When the *path* argument points to the string "F1/" and `F1` is a directory, then *mkdir*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

10(B) When the *path* argument points to the string "F1//" and `F1` is a directory, then *mkdir*() resolves the pathname to `F1`, which is in the current directory. (See GA20 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

11(A) When the *path* argument points to the string "F1/F2 ", then *mkdir*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the *path* argument points to the string " F1/./F2", then *mkdir*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string " F1/../ F1/F2", then *mkdir*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string " F1//F2 ", then *mkdir*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

When the *path* component is a string of more than {_NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *mkdir*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *mkdir(path, mode)*, the pathname *path* supports filenames containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *mkdir(path, mode)*, the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

18(A) When *mkdir(path, ...)* is granted write access to the parent directory of *path* and search access to the path prefix of *path*, then the standard file access control mechanism does not cause *mkdir()* to indicate that file access is denied.

19(A) A call to *mkdir(path, mode)* creates a new directory, with file permission bits determined by the *mode* argument and the file creation mask, and returns a value of zero.

D01(A) The details describing the effect on *mkdir()* of bits other than permission bits being set in *mode* are contained in 5.4.1.2 of the PCD.1.

20(A) The owner ID of the newly created directory is set to the effective user ID of the process.

D02(C) If the conditions under which the group ID of the new directory is set to the group ID of the directory in which the directory is being created and the conditions under which the group ID of the directory is set to the effective group ID of the calling process for a successful call to *mkdir()* are documented:
The details are contained in 5.4.1.2 of the PCD.1.

21(A) The group ID of the newly created directory is set to the effective group ID of the process or to the group ID of the directory in which the directory is being created.

22(A) The newly created directory is an empty directory.

23(A) A call to *mkdir()* marks for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the new directory and the *st_ctime* and *st_mtime* fields of the parent directory.

5.4.1.3 Returns

R01 When a call to *mkdir()* completes successfully, then a value of (*int*)0 is returned. (See Assertion 19 in 5.4.1.2.)

R02 When a call to *mkdir()* completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertions 24–29 and 31–36 in 5.4.1.4.)

5.4.1.4 Errors

24(A) When search permission is denied for a component of the path prefix of *path*, then a call to *mkdir(path, mode)* returns a value of (*int*)-1 and sets *errno* to [EACCES].

25(A) When write permission is denied on the parent directory of the directory to be created, then a call to *mkdir(path, mode)* returns a value of (*int*)-1 and sets *errno* to [EACCES].

26(A) When the file named by *path* exists, then a call to *mkdir(path, mode)* returns a value of(*int*)-1 and sets *errno* to [EEXIST].

27({LINK_MAX}≤{PCTS_LINK_MAX}?C:UNTESTED)

If creating a directory causes the link count of the directory in which *path* is to be created to be incremented:
When {LINK_MAX} links already exist in the directory in which *path* is to be created, then a call to *mkdir(path, mode)* returns a value of (*int*)-1, sets *errno* to [EMLINK], and does not create a directory.
Testing Requirements:
Test is to be performed in a directory with has no more than "." and ".." entries.

28({LINK_MAX}>{PCTS_LINK_MAX}?C:UNTESTED)

If creating a directory causes the link count of the directory in which *path* is to be created to be incremented:
{PCTS_LINK_MAX} directories can be created in one directory.
Testing Requirements:
Test is to be performed in a directory with has no more than "." and ".." entries.

29({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *mkdir(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

30({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS_NAMEMAX}, then a call to *mkdir* (*path, mode*) succeeds.

31(A) If {PATH_MAX} ≤ {PCTS_PATH_MAX}:

When the length of the path argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *mkdir(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

Otherwise:

When the length of the path argument is {PCTS_PATH MAX}, then a call to *mkdir*() is successful.

32(A) When a component of the path prefix of *path* does not exist, then a call to *mkdir(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

33(A) When the *path* argument points to an empty string, then a call to *mkdir(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

34(B) When the file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory, then a call to *mkdir(path, mode)* returns a value of (*int*)-1, sets *errno* to [ENOSPC], and does not create a directory.

See Reason 1 in Section 5. of POSIX.3 {4}.

35(A) When a component of the path prefix of *path* is not a directory, then a call to *mkdir(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOTDIR].

36(C) If the implementation supports a read-only file system:

When the parent directory resides on a read-only file system, then a call to *mkdir(path, mode)* returns a value of (*int*)-1, sets *errno* to [EROFS], does not create a directory, and does not mark for update the *st_atime, st_ctime,* and *st_mtime* fields of the parent directory. (See GA13 in 2.3.5.)

5.4.2 *mkfifo*()

5.4.2.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <sys/stat.h> is included, then the function prototype `int mkfifo(const char *, mode_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <sys/stat.h> is included, then the function *mkfifo()* is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *mkfifo()* is defined as a macro when the header <sys/stat.h> is included:

When the macro *mkfifo()* is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *mkfifo()* is defined as a macro in the header <sys/stat.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5.4.2.2 Description

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *mkfifo()* resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(B) When the *path* argument points to the string "/", then *mkfifo()* resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

06(B) When the *path* argument points to the string "/ / /", then *mkfifo()* resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *mkfifo()* resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *mkfifo()* resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(B) When the *path* argument points to the string "F1/" and `F1` is a directory, then *mkfifo()* resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

10(B) When the *path* argument points to the string "F1//" and `F1` is a directory, then *mkfifo()* resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

11(A) When the *path* argument points to the string "F1/F2", then *mkfifo()* resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the *path* argument points to the string "F1/./F2", then *mkfifo()* resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string "F1/../F1/F2", then *mkfifo()* resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string "F1//F2", then *mkfifo()* resolves the pathname to the file `F2` in the directory, `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:
When the *path* component is a string of more, than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *mkfifo()* resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *mkfifo(path, mode)*, the pathname *path* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *mkfifo(path, mode)*, the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18(A) When *mkfifo(path, ...)* is granted write access to the parent directory of *path* and search access to the path prefix of *path*, then the standard file access control mechanism does not cause *mkfifo()* to indicate that file access is denied.

19(A) A call to *mkfifo*() creates a new FIFO special file with file permission bits determined by the *mode* argument and the file creation mask, and returns a value of zero.

D01(A) The details describing the effect on *mkfifo*() of bits other than permission bits being set in *mode* are contained in 5.4.2.2 of the PCD.1.

20(A) The owner ID of the newly created FIFO is set to the effective user ID of the process.

D02(C) If the conditions under which the group ID of the new FIFO is set to the group ID of the directory in which the FIFO is being created and the conditions under which the group ID of the FIFO is set to the effective group ID of the calling process for a successful call to *mkfifo()* are documented:

The details are contained in 5.4.2.2 of the PCD.1.

21(A) The group ID of the newly created FIFO is set to the effective group ID of the process, or the group ID of the directory in which the FIFO is being created.

22(A) A call to *mkfifo*() marks for update the *st_atime*, *st_mtime*, and *st_ctime* fields of the FIFO file created and the *st mtime* and *st_ctime* fields of the parent directory of the FIFO file.

5.4.2.3 Returns

R01 When a call to *mkfifo*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 19 in 5.4.2.2.)

R02 When a call to *mkfifo*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertions 23–26 and 28–33 in 5.4.2.4.)

5.4.2.4 Errors

23(A) When search permission is denied for any component of the path prefix of *path*, then a call to *mkfifo(path, mode)* returns a value of (*int*)-1 and sets *errno* to [EACCES].

24(A) When write permission is denied on the parent directory of the FIFO to be created, then a call to *mkfifo(path, mode)* returns a value of (*int*)-1 and sets *errno* to [EACCES].

25(A) When the file named by *path* exists, then a call to *mkfifo(path, mode)* returns a value of (*int*)-1 and sets *errno* to [EEXIST].

26({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *mkfifo(path, mode)* returns a value of (*int*)-1, sets *errno* to [ENAMETOOLONG], and does not create a FIFO.

27({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS NAME_MAX}, then a call to *mkfifo(path, mode)* succeeds.

28(A) If {PATH_MAX}≤{PCTS_PATH_MAX}:

When the length of the *path* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *mkfifo(path, mode)* returns a value of (*int*)-1, sets *errno* to [ENAMETOOLONG], and does not create a FIFO.

Otherwise:

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the length of the *path* argument is {PCTS_PATH_MAX}, then a call to *mkfifo*() is successful.

29(A) When a component of the path prefix of *path* does not exist, then a call to *mkfifo(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

30(A) When the *path* argument points to an empty string, then a call to *mkfifo(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

31(B) When the directory that would contain the new file cannot be extended or the file system is out of file allocation resources, then a call to *mkfifo(path, mode)* returns a value of (*int*)-1, sets *errno* to [ENOSPC], and does not create a FIFO.

See Reason 1 in Section 5. of POSIX.3 {4}.

32(A) When a component of the path prefix of *path* is not a directory, then a call to *mkfifo(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOTDIR].

33(C) If the implementation supports a read-only file system:

When the named file resides on a read-only file system, then a call to *mkfifo(path, mode)* returns a value of (*int*)-1, sets *errno* to [EROFS], does not create a FIFO, and does not mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the parent directory. (See GA13 in 2.3.5.)

5.5 File Removal

5.5.1 *unlink*()

5.5.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `int unlink(const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *unlink*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *unlink*() is defined as a macro when the header <unistd.h> is included:

When the macro *unlink*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *unlink*() is defined as a macro in the header <unistd.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.5.1.2 Description

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *unlink*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(B) When the *path* argument points to the string "/", then *unlink*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

06(B) When the *path* argument points to the string "///", then *unlink*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *unlink*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *unlink*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(C) If {PCD_LINK_TO_DIRECTORY} is **TRUE**:
When the *path* argument points to the string "F1/" and `F1` is a directory, then *unlink*() resolves the pathname to Fl, which is in the current working directory. (See GA19 in 2.3.6.)

10(C) If {PCD_LINK_TO_DIRECTORY} is **TRUE:**
When the *path* argument points to the string "F1//" and `F1` is a directory, then *unlink*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *path* argument points to the string "F1/F2", then *unlink*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the *path* argument points to the string "F1/./F2", then *unlink*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string "F1/../F1/F2", then *unlink*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string "F1//F2", then *unlink*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:
When the *path* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *unlink*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *unlink(path)*, the pathname *path* supports *filenames* containing any of the following characters:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *unlink(path)*, the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) When *unlink(path)* is granted write access to the parent directory of *path* and search access to the path prefix of path, then the standard file access control mechanism does not cause *unlink*() to indicate that file access is denied.

19(A) A call to *unlink(path)* removes the link named by *path*, decrements the link count of the file referenced by the link, and returns a value of zero.

20(B) When the link count becomes zero and the file is not opened by a process, then the space occupied by the file is freed.

See Reason 1 in Section 5. of POSIX.3 [4].

21(B) When the link count becomes zero and the file is not, opened by a process, then the file is no longer accessible.

See Reason 3 in Section 5. of POSIX.3 [4].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22(A) When one or more processes have the file open when the last link is removed, then the link is removed before *unlink*() returns, but the removal of the file contents is postponed until all references to the file have been closed.

D01(C) If the implementation supports a method to associate the appropriate privileges for unlinking directories with a process:

The details are documented in 2.2.2.4 or in 5.5.1.2 of the PCD.1.

D02(C) If the implementation supports *unlink*() on directories, and this is documented:

The details are contained in 5.5.1.2 of the PCD.1.

23(D) If the implementation supports unlinking a directory:

When the calling process has the appropriate privileges to unlink a directory, then a call to *unlink*() can unlink a directory.

See Reason 1 in Section 5. of POSIX.3 [4].

24(A) A successful call to *unlink*() marks for update the *st_ctime* and *st_mtime* fields of the parent directory.

25(A) When the link count of the file does not become zero, s successful call to *unlink*() marks for update the *st_ctime* field of the file.

26(D) If the implementation supports unlinking a directory and the implementation does not consider unlinking the directory named by the *path* argument an error when it is being used by the system or, another process:

When the calling process has the appropriate privileges to unlink a directory, and when the directory named by the *path* argument is being used by the system or another process. then a call to *unlink(path)* successfully unlinks the directory *path.*

See Reason 2 in Section 5. of POSIX.3 [4].

5.5.1.3 Returns

R01 When a call to *unlink*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 19 in 5.5.1.2.)

R02 When a call to *unlink*() completes unsuccessfully, then a value of (*int*)–1 is returned and sets *errno* to indicate the error. (See Assertions 27–30 and 32–39 in 5.5.1.4.)

5.5.1.4 Errors

27(A) When search permission is denied for a component of the path prefix of *path*, then a call to *unlink(path)* returns a value of (*int*)-1, sets *errno* to [EACCES], and does not unlink the directory entry.

28(A) When write permission is denied on the directory containing the link to be removed, then a call to *unlink(path)* returns a value of (*int*)-1, sets *errno* to [EACCES], and does not unlink the directory entry.

D03(C) If the implementation supports the error condition [EBUSY] for *unlink*() when the directory named by the path argument cannot be unlinked because it is being used by the system or another process, and this is documented:

The details are contained in 5.5.1.4 of the PCD.1.

29(D) If the implementation supports unlinking a directory and the implementation considers an attempt to unlink a directory chat is being used by the system or another process to be an [EBUSY] error:

When the calling process has the appropriate privileges to unlink a directory, and when the directory named by the *path* argument cannot be unlinked because it is being used by the system or another process, then a call to *unlink(path)* returns a value of (*int*)-1 and sets *errno* to [EBUSY].

See Reason 2 in Section 5. of POSIX.3 [4].

30({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860276 759 ■

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *unlink(path)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

31({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS_NAME_MAX}, then a call to *unlink(path)* succeeds.

32(A) If {PATH_MAX}≤{PCTS_PATH_MAX}:

When the length of the *path* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *unlink(path)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

Otherwise:

When the length of the path argument is {PCTS_PATH_MAX}, then a call to *unlink*() is successful.

33(A) When a component of the path prefix of *path* does not exist, then a call to *unlink(path)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

34(A) When the file named by *path* does not exist, then a call to *unlink(path)* returns a value of (*int*)-1 and sets *errno* to {ENOENT}.

35(A) When the *path* argument points to an empty string, then a call to *unlink(path)* returns a value of (*int*)-1 and sets *errno* to {ENOENT}.

36(A) When a component of the path prefix of *path* is not a directory, then a call to *unlink(path)* returns a value of (*int*)-1 and sets *errno* to {ENOTDIR}.

37(C) If the implementation supports the linking and unlinking of directories:

When the calling process does not have the appropriate privileges to unlink a directory and *path* is a directory, then a call to *unlink(path)* returns a value of (*int*)-1, sets *errno* to [EPERM], and does not unlink the directory.

38(D) If the implementation does not support the unlinking of directories:

When *path* refers to a directory, then a call to *unlink(path)* returns a value of (*int*)-1, sets *errno* to {EPERM}, and does not unlink the directory.

See Reason 1 in Section 5. of POSIX.3 [4].

39(C) If the implementation supports a read-only file system:

When the named file resides on a read-only file system, then a call to *unlink(path)* returns a value of (*int*)-1, sets *errno* to {EROFS}, does not unlink the file entry, and does not mark for update the *st_atime, st_ctime,* and *st_mtime* fields of the parent. directory of the file. (See GA13 in 2.3.5.)

5.5.2 *rmdir*()

5.5.2.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `int rmdir(const: char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *rmdir*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *rmdir*() is defined as a macro when the header <unistd.h> is included:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When the macro *rmdir*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *rmdir*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.5.2.2 Description

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *rmdir*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

D01(C) If a call to *rmdir(path)* succeeds when the argument path is the root directory or the current directory of any process, and this is documented:
The details are contained in 5.5.2.2 of the PCD.1.

D02(C) If a call to *rmdir(path)* fails and sets *errno* to [EBUSY] when the argument path is the root directory or the current directory of any process, and this is documented:
The details are contained in 5.5.2.2 of the PCD.1.

05(B) When the *path* argument points to the string "/", then *rmdir*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

See Reason 7 in Section 5. of POSIX.3 {4}.

06(B) When the *path* argument points to the string "////", then *rmdir*() resolves the pathname to the root directory of the process. (,See GA16 in 2.3.6.)

See Reason 7 in Section 5. of POSIX.3 {4}.

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *rmdir*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *rmdir*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *path* argument points to the string "F1/" and `F1` is a directory, then *rmdir*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

10(A) When the *path* argument points to the string "F1//" and `F1` is a directory, then *rmdir*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *path* argument points to the string "F1/F2", then *rmdir*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the pathname argument points to the string "F1/./F2", then *rmdir*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string "F1/../F1/F2", then *rmdir*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string "F1//F2", then *rmdir*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When the *path* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported,then *rmdir*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *rmdir(path)*, the pathname *path* supports *filenames* containing any of the following characters:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *rmdir(path)*, the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) When *rmdir(path)* is granted write access to the parent directory of *path* and search access to the *path* prefix of path, then the standard file access control mechanism does not cause *rmdir*() to indicate that file access is denied.

19(A) A call to *rmdir*() removes the empty directory named by *path* and returns a value of zero.

20(B) When the link count of the directory being removed becomes zero and no process has the directory open, then the space occupied by the directory is freed and the directory is no longer accessible.

See Reason 1 in Section 5. of POSIX.3 [4]

21(A) When a process has the directory open and a call to *rmdir*() removes the last link to the directory, then when *rmdir*() returns there are no dot and dot-dot entries in the directory, and no new entries may be created in the directory.

22(B) When a process has the directory open when the last link is removed, then the directory is not removed until all references to the directory have been closed.

See Reason 1 in Section 5. of POSIX.3 [4].

23(A) A successful call to *rmdir*() marks for update the *st_ctime* and *st_mtime* fields of the parent directory.

24(D) If the implementation allows the removal of a directory that is being used by another process:
When the directory named by the *path* argument is being used by another process, then a call to *rmdir(path)* removes the directory and returns successfully.

See Reason 2 in Section 5. of POSIX.3 [4].

5.5.2.3 Returns

R01 When a call to *rmdir*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 19 in 5.5.2.2.)

R02 When a call to *rmdir*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error, and the directory is not removed. (See Assertions 25–29 and 31–36 in 5.5.2.4.)

5.5.2.4 Errors

25(A) When search permission is denied for a component of the path prefix of *path*, then a call to *rmdir(path)* returns a value of (*int*)-1, sets *errno* to [EACCES], and does not remove the directory.

26(A) When write permission is denied on the parent directory of the directory to be removed, then a call to *rmdir(path)* returns a value of (*int*)-1, sets *errno* to [EACCES], and does not remove the directory.

D03(C) If the implementation supports the error condition [EBUSY] for *rmdir*() when the directory named by the path argument cannot be removed because it is being used by another process:
The details are contained in 5.5.2.4 of the PCD.1. (See DGA02 in 2.4.)

27(D) If the implementation does not allow the removal of a directory that is being used by another process:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the directory named by the *path* argument cannot be removed because it is being used by another process, and when the implementation considers this to be an error, then a call to *rmdir(path)* returns a value of (*int*)-1, sets *errno* to [EBUSY], and does not remove the directory.

See Reason 2 in Section 5. of POSIX.3 {4}.

28(A) When the *path* argument names a directory that is not an empty directory, then a call to *rmdir(path)* returns a value of (*int*)-1, sets *errno* to [EEXIST] or [ENOTEMPTY], and does not remove the directory.

29({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO TRUNC} is supported for the file:

When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *rmdir(path)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

30({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS_NAME_MAX}, then a call to *rmdir(path)* succeeds.

31(A) If {PATH_MAX} ≤ {PCTS_PATH_MAX}

When the length of the *path* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *rmdir(path)* returns a value of (*int*)-1, sets *errno* to [ENAMETOOLONG], and does not remove the directory.

Otherwise:

When the length of the *path* argument is [PCTS_PATH_MAX], then a call to *rmdir*() is successful.

32(A) When a component of the path prefix of *path* does not exist, then a call to *rmdir(path)* returns a value of(*int*)-1 and sets *errno* to [ENOENT].

33(A) When the *path* argument names a nonexistent directory, then a call to *rmdir(path)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

34(A) When the *path* argument points to an empty string, then a call to *rmdir(path)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

35(A) When a component of *path* is not a directory, then a call to *rmdir(path)* returns a value of(*int*)-1 and sets *errno* to [ENOTDIR].

36(C) If the implementation supports a read-only file system:

When the directory entry to be removed resides on a read-only file system, then a call to *rmdir(path)* returns a value of (*int*)-1, sets *errno* to [EROFS], does not remove the named directory entry, and does not mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the parent directory of the named directory entry. (See GA13 in 2.3.5.)

5.5.3 *rename*()

5.5.3.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype `int rename(const char *, const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *rename*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *rename*() is defined as a macro when the header <stdio.h> is included:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the macro *rename*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *rename*() is defined as a macro in the header `<stdio.h>`:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.5.3.2 Description

04(A) When the first filename component of the *old* argument is ".", and the pathname does not begin with a slash, then *rename*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(B) When the *old* argument points to the string "/", then *rename*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

06(B) When the *old* argument points to the string "///", then *rename*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

07(A) When the *old* argument points to a string beginning with a single slash or beginning with three or more slashes, then *rename*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *old* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *rename*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *old* argument points to the string "F1/" and `F1` is a directory, then *rename*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

10(A) When the *old* argument points to the string "F1//" and `F1` is a directory, then *rename*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *old* argument points to the string "F1/F2", then *rename*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the *old* argument points to the string "F1/./F2", then *rename*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *old* argument points to the string "F1/../F1/F2", then *rename*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *old* argument points to the string "F1//F2", then *rename*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:
When the old component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *rename*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *rename*(*old, new*), the pathname *old* supports *filenames* confining any of the following characters:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17(A) On a call to *rename(old, new)*, the pathname *old* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) When the first filename component of the *new* argument is ".", the argument points to a string beginning with a single slash or beginning with three or more slashes, then *rename*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

19(B) When the *new* argument points to the string "/", then *rename*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

20(B) When the *new* argument points to the string "///", then *rename*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

21(A) When the *new* argument points to a string beginning with a slash, then *rename*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

22(A) When the first filename component of the *new* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *rename*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

23(A) When the *new* argument points to the string "F1/" and `F1` is an empty directory, then *rename*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

24(A) When the *new* argument points to the string "F1//" and F1 is an empty directory, then *rename*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

25(A) When the *new* argument points to the string "F1/F2", then *rename*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

26(A) When the *new* argument points to the string "F1/./F2", then *rename*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

27(A) When the *new* argument points to the string "F1/../F1/F2", then *rename*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

28(A) When the *new* argument points to the string "F1//F2", then *rename*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

29(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

When the *new* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *rename*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

30(A) On a call to *rename(old, new)*, the pathname *new* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

31(A) On a call to *rename(old, new)*, the pathname *new* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

32(A) When *rename(old, new)* is granted write access to the parent directories of both old and new and search access to the path prefixes of both *old* and *new*, then the standard file access control mechanism does not cause *rename*() to indicate that file access is denied.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

33(A) When old and new do not refer to the same existing file, then after a successful call to *rename(old, new)* the old pathname no longer exists, and a value of zero is returned.

34(A) When the *old* and the *new* arguments both refer to links to the same existing file, then a call to *rename(old, new)* returns successfully without taking any other action.

35(B) When *new* refers to an existing file, then a link named by argument *new* exists throughout the renaming operation and refers either to the file referred to by argument *new* or *old* before the operation began.

See Reason 1 in Section 5. of POSIX.3 [4].

36(A) When the link named by the *new* argument exists, then a call to *rename(old, new)* removes *new* and renames *old* to *new*.

R01 A call to *rename(old, new)* requires write access permission for both the directory containing *old* and the directory containing *new*. (See Assertion 45 in 5.5.3.4.)

37(A) When the *old* argument points to the pathname of a directory, and when the directory named by the *new* argument exists and is empty, then a call to *rename(old, new)* removes *new* and renames *old* to *new*.

R02 When the *new* argument names an existing directory, then it is required to be an empty directory. (See Assertion 49 in 5.5.3.4.)

38(B) When the link named by the *new* argument exists, no process has the file open, and the link count of the file becomes zero, then a call to *rename(old, new)* frees the space occupied by the file.

See Reason 1 in Section 5. of POSIX.3 [4].

39(A) When one or more processes have the file argument new open and the last link is removed, then a call to *rename(old, new)* removes the link before the *rename*() returns, but the removal of the file contents is postponed until all references to the file have been closed.

D01(C) If it is documented whether the implementation requires write access for the directory named by *old* and, if it exists, for the directory named by *new*, for a successful *rename(old, new)* call:

The details are contained in 5.5.3.2 of the PCD.1.

40(C) If {PCD_WRITE_PERM_TO_RENAME} is **FALSE**:

When write permission is denied for a directory pointed to by the *old* or *new* arguments, then a call to *rename(old, new)* completes successfully.

R03 When write permission is denied for a directory pointed to by the *old* or *new* arguments, then a call to *rename(old, new)* either completes uccessfully or returns a value of (*int*)-1, sets *errno* to [EACCES], and does not change the named files. (See Assertions 46 and 47 in 5.5.3.4.)

41(D) If the implementation supports a call to *rename(old, new)* when the directory named by *old* or *new* is being used by the system or another process:

When either *old* or *new* is being used by the system or another process, then a call to *rename(old, new)* successfully renames the directory old to new.

See Reason 2 in Section 5. of POSIX.3 [4].

42(C) If {PCD_LINK_FILE_SYSTEM} is **TRUE**:

When the links named by *old* and *new* are on different file systems, then a call to *rename(old, new)* completes successfully.

R04 When the links named by *old* and *new* are on different file systems, then a call to *rename(old, new)* is either successful or returns a value of *(int)–1*, sets *errno* to [EXDEV], and does not change the named files. (See Assertions 68 and 69 in 5.5.3.4.)

43(A) On a call to *rename(old, new)*, *rename*() marks for update the *st_ctime* and *st_mtime* fields of the parent directory of each file.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5.5.3.3 Returns

R05 When a call to *rename*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 33 in 5.5.3.2.)

R06 When a call to *rename*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the *errno*, and the named files are not changed. (See Assertions 44–54, 56–61, and 63–68 in 5.5.3.4.)

5.5.3.4 Errors

44(A) When the path prefix of either *old* or *new* denies search permission, then a call to *rename*(*old, new*) returns a value of (*int*)-1, sets *errno* to [EACCES], and does not change the named files.

45(A) When one of the directories containing *old* or *new* denies write permission, then a call to *rename*(*old, new*) returns a value of (*int*)-1, sets *errno* to [EACCES], and does not change the named files.

46(C) If {PCD_WRITE_PERM_TO_RENAME} is **TRUE**:
When write permission is denied for a directory pointed to by the *old* or *new* arguments, then a call to *rename*(*old, new*) returns a value of (*int*)-1, sets *errno* to [EACCES], and does not change the named files.

47(C) If {PCD WRITE_PERM_TO_RENAME} is not documented:
When write permission is denied for a directory pointed to by the *old* or *new* arguments, then a call to *rename*(*old, new*) either completes successfully or returns a value of (*int*)-1, sets *errno* to [EACCES], and does not change the named files.

D02(C) If the implementation supports the error condition [EBUSY] for *rename*(*old, new*) when the directory *old* or *new* cannot be renamed because it is being used by another process:
The details are contained in 5.5.3.4 of the PCD.1.

48(D) If the implementation considers a call to *rename*(*old, new*) when the directory named by *old* or *new* is being used by the system or another process to be an error:
When the directory named by *old* or *new* is being used by the system or another process, then a call to *rename*(*old, new*) returns a value of (*int*)-1, sets *errno* to [EBUSY], and does not change the named files.

See Reason 2 in Section 5. of POSIX.3 {4}.

49(A) When the link named by *new* is a directory containing entries other than dot and dot-dot, then a call to *rename*(*old, new*) returns a value of (*int*)-1, sets *errno* to [EEXIST] or [ENOTEMPTY], and does not change the named files.

50(A) When *new* and *old* name existing directories and the *new* pathname contains a path prefix that refers to old, then a call to *rename*(*old, new*) returns a value of (*int*)-1, sets *errno* to [EINVAL], and does not change the named files.

51(A) When the *new* argument points to a directory and the *old* argument points to a file that is not a directory, then a call to *rename*(*old, new*) returns a value of(*int*)-1, sets *errno* to [EISDIR], and does not change the named files.

52({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:
When the length of a pathname component of the *old* argument is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *rename*(*old, new*) returns a value of (*int*)-1, sets *errno* to [ENAMETOOLONG], and does not change the existing files.

53({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the length of a pathname component of the *new* argument is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *rename(old, new)* returns a value of (*int*)-1, sets *errno* to [ENAMETOOLONG], and does not change *old*.

54({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of the *old* argument equals {PCTS_NAME_MAX}, then a call to *rename(old, new)* succeeds.

55({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of the *new* argument equals {PCTS_NAME_MAX}, then a call to *rename(old, new)* succeeds.

56(A) If {PATH_MAX}≤{PCTS_PATH_MAX}:

When the length of the *old* argument exceeds the maximum number of bytes in a pathname {PATHMAX}, then a call to *rename(old, new)* returns a value of(*int*)-1, sets *errno* to [ENAMETOOLONG], and does not change the existing files.

Otherwise:

When the length of the *old* argument is {PCTS_PATH_MAX}, then a call to *rename*() is successful.

57(A) If {PATH_MAX}≤{PCTS_PATH_MAX}:

When the length of the *new* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *rename(old, new)* returns a value of (*int*)-1, sets *errno* to [ENAMETOOLONG], and does not change the *old* file.

Otherwise:

When the length of the *new* argument is {PCTS_PATH_MAX}, then a call to *rename*() is successful.

58(A) When a component of one of the path prefixes *old* or *new* does not exist, then a call to *rename(old, new)* returns a value of (*int*)-1, sets *errno* to [ENOENT], and does not change the existing files.

59(A) When the link named by the *old* argument does not exist, then a call to *rename(old, new)* returns a value of (*int*)-1, sets *errno* to [ENOENT], and does not change the existing files.

60(A) When either the *old* or *new* argument points to an empty string, then a call to *rename(old, new)* returns a value of (*int*)−1, sets *errno* to [ENOENT], and does not change the existing files.

61({LINK_MAX}≤{PCTS_LINK_MAX}?C:UNTESTED)

If {PCD_CREAT_LINK_COUNT} is **TRUE:**

When the file named by *old* is a directory and {LINK_MAX} links already exist to the directory in which *new* is to be created, then a call to *rename(old, new)* returns a value of (*int*)-1 and sets *errno* to [EMLINK].

62({LINK_MAX}≤{PCTS_LINK_MAX}?C:UNTESTED)

If {PCD_CREAT_LINK_COUNT} is **FALSE:**

When the file named by *old* is a directory and {LINK_MAX} links already exist to the directory in which *new* is to be created, then a call to *rename(old, new)* succeeds.

63({LINK_MAX}≤{PCTS_LINK_MAX}?C:UNTESTED)

If {PCD_CREAT_LINK COUNT} is not documented:

When the file named by *old* is a directory and {LINK_MAX} links already exist to the directory in which *new* is to be created, then a call to *rename(old, new)* either succeeds or returns a value of (*int*)-1 and sets *errno* to [EMLINK].

64(B) When the directory that would contain *new* cannot be extended, then a call to *rename(old, new)* returns a value of (*int*)-1, sets *errno* to [ENOSPC], and does not change the named files.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860285 761 ■

See Reason 1 in Section 5. of POSIX.3 {4}.

65(A) When a component of the path prefix of either *old* or *new* is not a directory, then a call to *rename(old, new)* returns a value of (*int*)-1, sets *errno* to [ENOTDIR], and does not change the named files.

66(A) When the *old* argument names a directory and the *new* argument names a nondirectory, then a call to *rename(old, new)* returns a value of (*int*)-1, sets *errno* to [ENOTDIR], and does not change the named files.

67(C) If the implementation supports a read-only file system:
When the requested operation requires writing in a directory on a read-only file system, then a call to *rename(old, new)* returns a value of (*int*)-1, sets *errno* to [EROFS], does not change the named files, and does not mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the parent directories of the named files. (See GA13 in 5.2.3.)

68(C) If {PCD_LINK_FILE_SYSTEM} is **FALSE**:"
When the links named by *old* and *new* are on different file systems, then a call to *rename(old, new)* returns a value of (*int*)-1, sets *errno* to [EXDEV], and does not change the named files.

69(C) If {PCD_LINK_FILE_SYSTEM} is not documented:
When the links named by *old* and *new* are on different file systems, then a call to *rename(old, new)* is either successful or returns a value of (*int*)-1, sets *errno* to [EXDEV], and does not change the named files.

5.6 File Characteristics

5.6.1 File Characteristics: Header and Data Structure

01(A) The following elements are defined in *struct stat* with the types as specified in Table 5.1.

Table 5.1—*stat* Structure

Element	Type
st_mode	mode_t
st_ino	ino_t
st_dev	dev_t
st_nlink	nlink_t
st_uid	uid_t
st_gid	gid_t
st_size	off_t
st_atime	time_t
st_mtime	time_t
st_ctime	time_t

02(A) When the header <sys/stat.h> is included, then the *struct stat* is defined.

GA51 For *stat*() and *fstat*():
The *st_mode* member of the stat structure contains the file mode associated with the file referenced.

GA52 For *stat*() and *fstat*():
The *st_ino* and *st_dev* members of the stat structure taken together provide a unique reference to the file.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

GA53 For *stat*() and *fstat*():
The *st_nlink* member of the stat structure contains the number of links associated with the file referenced.

GA54 For *stat*() and *fstat*():
The *st_uid* member of the stat structure contains the user ID associated with the owner of the file referenced.

GA55 For *stat*() and *fstat*():
The *st_gid* member of the stat structure contains the group ID associated with the group owner of the file referenced.

D01(C) If the usage of the field *st_size* in the structure returned by *stat*() and *fstat*() for nonregular files is documented:
The details are contained in 5.6.1 of the PCD.1.

GA56 For *stat*() and *fstat*():
The *st_size* member of the stat structure contains the file size associated with the file referenced.

GA57 For *stat*() and *fstat*():
The *st_atime* member of the stat structure contains the time of last access associated with the file referenced.

GA58 For *stat*() and *fstat*():
The *st_mtime* member of the stat structure contains the time of last data modification associated with the file referenced.

GA59 For *stat*() and *fstat*():
The *st_ctime* member of the stat structure contains the time of last file status change associated with the file referenced.

5.6.1.1 <sys/stat.h> File Types

03(A) When the argument *m* (the *st_mode* member of *struct stat*) refers to a directory file, then the macro *S_ISDIR(m)* evaluates to nonzero. When the argument *m* refers to another file type, then the macro *S_ISDIR(m)* evaluates to zero.

04(C) If the implementation supports character special files:
When the argument *m* (the *st_mode* member of *struct stat*) refers to a character special file, then the macro *S_ISCHR(m)* evaluates to nonzero. When the argument *m* refers to another file type, then the macro *S_ISCHR(m)* evaluates to zero.

05(C) If the implementation supports block special files:
When the argument *m* (the *st_mode* member of *struct stat*) refers to a block special file, then the macro *S_ISBLK*(*m*) evaluates to nonzero. When the argument *m* refers to another file type, then the macro *S_ISBLK*(*m*) evaluates to zero.

06(A) When the argument *m* (the *st_mode* member of *struct star*) refers to a regular file, then the macro *S_ISREG(m)* evaluates to nonzero. When the argument m refers to another file type, then the macro *S_ISREG(m)* evaluates to zero.

07(A) When the argument *m* (the *st_mode* member of *struct star*) refers to a pipe or FIFO file, then the macro *S_ISFIFO(m)* evaluates to nonzero. When the argument *m* refers to another file type, then the macro *S_ISFIFO(m)* evaluates to zero.

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5.6.1.2 <sys/stat.h> File Modes

08(A) When the header <sys/stat.h> is included, then the mode value for symbols S_IRWXU, S_IRUSR, S_IWUSR, S_IXUSR, S_IRWXG, S_IRGRP, S_IWGRP, S_IXGRP, S_IRWXO, S_IROTH, S_IWOTH, S_IXOTH, S_ISUID, and S_ISGID are defined.

09(A) When the file mode is specified by the symbol as shown in Table 5.2, the stated permission is provided for the file owner class.

10(A) When the file mode is specified by the symbol as shown in Table 5.3, the stated permission is provided for the file group class.

11(A) When the file mode is specified by the symbol as shown in Table 5.4, the stated permission is provided for the file other class.

Table 5.2—File Owner Class Permission Masks

Symbol	Permission
S_IRUSR	read
S_IWUSR	write
S_IXUSR	execute

Table 5.3—File Group Class Permission Masks

Symbol	Permission
S_IRGRP	read
S_IWGRP	write
S_IXGRP	execute

Table 5.4—File Other Class Permission Masks

Symbol	Permission
S_IROTH	read
S_IWOTH	write
S_IXOTH	execute

12(A) The symbols S_IRWXU, S_IRWXG, and S_IRWXO are masks for read, write, and search (for a directory) or execute (for other file types) permissions for the file owner, group, and other class respectively.

13(A) When the S_ISUID mask is set on a file and the file is executed by one of the exec type functions, then the effective user ID of the process is set to that of the owner of the file.

14(A) When the S_ISGID mask is set on a file and the file is executed by one of the exec type functions, then the effective group ID of the process is set to that of the group owner of the file.

15(A) The values of the symbols S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH, S_ISUID, and S_ISGID are bitwise discrete.

16(A) The bits defined by the symbol S_IRWXU are the bitwise inclusive OR of S_IRUSR, S_IWUSR, and S_IXUSR.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17(A) The bits defined by the symbol S_IRWXG are the bitwise inclusive OR of S_IRGRP, S_IWGRP, and S_IXGRP.

18(A) The bits defined by the symbol S_IRWXO are the bitwise inclusive OR of S_IROTH, S_IWOTH, and S_IXOTH.

D02(C) If the implementation ORs bits other than those in S_IRWXU, S_IRWXG, and S_IRWXO into the *st_mode* field of the structure returned by *stat*() and *fstat*():
The details are contained in 5.6.1.2 of the PCD.1.

5.6.1.3 <sys/stat.h> Time Entries

19(B) The time-related fields are measured in seconds since the Epoch (00:00:00, January 1, 1970 Coordinated Universal Time).

See Reason 1 in Section 5. of POSIX.3 [4].

5.6.2 Get File Status

5.6.2.1 *stat*() (5.6.2)

5.6.2.1.1 Synopsis (5.6.2.1)

01(A) If the implementation provides C Standard {2} support:
When the header <sys/stat.h> is included, then the function prototype
`int stat(const char *, struct stat *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <sys/stat.h> is included, then the function *stat*() is either declared with result type *int* not declared in the header. (See GA36 in 2.7.3.)

02(C) If *stat*() is defined as a macro when the header <sys/stat.h> is included:
When the macro *stat*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *stat*() is defined as a macro in the header <sys/stat.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.6.2.1.2 Description (5.6.2.2)

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *stat*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(A) When the *path* argument points to the string "/", then *stat*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

06(A) When the *path* argument points to the string "///", then *stat*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *stat*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *stat*() resolves the pathname by

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *path* argument points to the string "F1/" and `F1` is a directory, then *stat*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

10(A) When the argument *path* points to the string "F1//" and `F1` is a directory, then *stat*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *path* argument points to the string "F1/F2", then *stat*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the *path* argument points to the string "F1/./F2", then *stat*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string "F1/../FI,F2", then *stat*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string "F1//F2", then *stat*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:
When the *path* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *stat*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *stat(path, buf)*, the pathname *path* supports *filenames* containing any of the following characters:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *stat(path, buf)*, the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) A successful call to *stat(path, buf)* stores, in the *st_mode* member of the *struct stat* pointed to by *buf*, the file mode associated with *path* and returns a value of zero. (See GA51 in 5.6.1.)

19(A) A successful call to *stat(path, buf)* stores, in the *st_ino* and *st_dev* members of the *struct stat* pointed to by *buf*, a unique reference to the file—when *st_ino* and *st_dev* are taken together—and returns a value of zero. (See GA52 in 5.6.1.)

20(A) A successful call to *stat(path, buf)* stores, in the *st_nlink* member of the *struct stat* pointed to by *buf*, the number of links associated with the file referenced and returns a value of zero. (See GA53 in 5.6.1.)

21(A) A successful call to *stat(path, buf)* stores, in the *st_uid* member of the *struct stat* pointed to by *buf*, the user ID associated with the owner of the file referenced and returns a value of zero. (See GA54 in 5.6.1.)

22(A) A successful call to *stat(path, buf)* stores, in the *st_gid* member of the *struct stat* pointed to by *buf*, the group ID associated with the group owner of the file referenced and returns a value of zero. (See GA55 in 5.6.1.)

23(A) A successful call to *stat(path, buf)* stores, in the *st_size* member of the *struct stat* pointed to by *buf*, the file size associated with the file referenced and returns a value of zero. (See GA56 in 5.6.1.)

24(A) A successful call to *stat(path, buf)* stores, in the *st_atime* member of the *struct stat* pointed to by *buf*, the time of last access associated with the file referenced and returns a value of zero. (See GA57 in 5.6.1.)

25(A) A successful call to *stat(path, buf)* stores, in the *st_mtime* member of the *struct stat* pointed to by *buf*, the time of last data modification associated with the file referenced and returns a value of zero. (See GA58 in 5.6.1.)

26(A) A successful call to *stat(path, buf)* stores, in the *st_ctime* member of the *struct stat* pointed to by *buf*, the time of last file status change associated with the file referenced and returns a value of zero. (See GA59 in 5.6.1.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

27(A) When *stat(path, bur)* is granted search access to the path prefix of *path,* then the standard file access control mechanism does not cause *stat*() to indicate that file access is denied.

R01 A successful call to *stat(path, buf)* stores a struct stat containing information about the file named by *path* in the area pointed to by *buf* and returns a value of zero. (See Assertions 18–26 in 5.6.2.1.2.)

28(A) A call to *stat*() does not need read, write, or execute permissions to the named file.

D01(C) If additional or alternate file access control mechanisms cause the *stat*() function to fail in ways not specified by POSIX.1 {3}:

The details when additional or alternate file access control mechanisms cause the *stat*() function to fail are contained in 5.6.2.2 of the PCD.1.

29(A) A call to *stat(path, buf)* updates any time-related fields marked for update for the file referenced by *path* before writing into the *struct star* and does not update time-related fields not marked for update. (See GA12 in 2.3.5.)

5.6.2.1.3 Returns (5.6.2.3)

R02 When a call to *stat*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 18–26 in 5.6.2.1.2.)

R03 When a call to *stat*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertions 30, 31 and 33–37 in 5.6.2.1.4.)

5.6.2.1.4 Errors (5.6.2.4)

30(A) When search permission is denied for a component of the path prefix of *path,* then a call to *stat(path, bud)* returns a value of (*int*)-1 and sets *errno* to [EACCES].

31({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *stat(path, buf)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

32({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS_NAME_MAX}, then a call to *stat(path, buf)* succeeds.

33(A) If {PATH_MAX} ≤ {PCTS_PATH_MAX}:

When the length of the path argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *stat(path, bud)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

Otherwise:

When the length of the *path* argument is {PCTS_PATH_MAX}, then a call to *stat*() is successful.

34(A) When the file named by *path* does not exist, then a call to *stat(path, buf)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

35(A) When a component of the path prefix of *path* does not exist, then a call to *stat(path, buf)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

36(A) When the *path* argument points to an empty string, then a call to *stat(path, buf)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

37(A) When the component of the path prefix of *path* is not a directory, then a call to *stat(path, bud)* returns a value of (*int*)-1 and sets *errno* to [ENOTDIR].

5.6.2.2 *fstat*() (5.6.2)

5.6.2.2.1 Synopsis (5.6.2.1)

01(A) If the implementation provides C Standard {2} support:

When the header `<sys/stat.h>` is included, then the function prototype `int fstat(int, struct stat *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header `<sys/stat.h>` is included, then the function *fstat*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *fstat*() is defined as a macro when the header `<sys/stat.h>` is included:

When the macro *fstat*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *fstat*() is defined as a macro in the header `<sys/stat.h>`:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.6.2.2.2 Description (5.6.2.2)

04(A) A successful call to *fstat(fildes, buf)* stores, in the *st_mode* member of the *struct stat* pointed to by *buf*, the file mode associated with *fildes* and returns a value of zero. (See GA51 in 5.6.1.)

05(A) A successful call to *fstat(fildes, buf)* stores, in the *st_ino* and *st_dev* members of the *struct stat* pointed to by *buf*, a unique reference to the file—when *st_ino* and *st_dev* are taken together—and returns a value of zero. (See GA52 in 5.6.1.)

06(A) A successful call to *fstat(fildes, buf)* stores, in the *st_nlink* member of the *struct stat* pointed to by *buf*, the number of links associated with the file referenced and returns a value of zero. (See GA53 in 5.6.1.)

07(A) A successful call to *fstat(fildes, buf)* stores, in the *st_uid* member of the *struct stat* pointed to by *buf*, the user ID associated with the owner of the file referenced and returns a value of zero. (See GA54 in 5.6.1.)

08(A) A successful call to *fstat(fildes, buf)* stores, in the *st_gid* member of the *struct stat* pointed to by *buf*, the group ID associated with the group owner of the file referenced and returns a value of zero. (See GA55 in 5.6.1.)

09(A) A successful call to *fstat(fildes, buf)* stores, in the *st_size* member of the *struct stat* pointed to by *buf*, the file size associated with the file referenced and returns a value of zero. (See GA56 in 5.6.1.)

10(A) A successful call to *fstat(fildes, buf)* stores, in the *st_atime* member of the *struct stat* pointed to by *buf*, the time of last access associated with the file referenced and returns a value of zero. (See GA57 in 5.6.1.)

11(A) A successful call to *fstat(fildes, buf)* stores, in the *st_mtime* member of the *struct stat* pointed to by *buf*, the time of last data modification associated with the file referenced and returns a value of zero. (See GA58 in 5.6.1.)

12(A) A successful call to *fstat(fildes, buf)* stores, in the *st_ctime* member of the *struct stat* pointed to by *buf*, the time of last file status change associated with the file referenced and returns a value of zero. (See GA59 in 5.6.1.)

R01 A call to *fstat(fildes, buf)* stores a *struct stat* containing information about the file associated with the named file descriptor in the area pointed to by *buf* and returns a value of zero. (See Assertions 4–12 in 5.6.2.2.2.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D01(C) If additional or alternate file access control mechanisms cause the *fstat*() function to fail in ways not specified by POSIX.1 {3}:

The details when additional or alternate file access control mechanisms cause the *fstat*() function to fail are contained in 5.6.2.2 of the PCD.1.

13(A) A call to *fstat(fildes, buf)* updates any time-related fields marked for update for the file referenced by *fildes* before writing into the *struct stat* and does not update time-related fields not marked for update. (See GA12 in 2.3.5.)

5.6.2.2.3 Returns (5.6.2.3)

R02 When a call to *fstat*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 4–12 in 5.6.2.2.2.)

R03 When a call to *fstat*() completes unsuccessfully, then a value of (*int*)–1 is returned and sets *errno* to indicate the error. (See Assertion 14 in 5.6.2.2.4.)

5.6.2.2.4 Errors (5.6.2.4)

14(A) When the file descriptor argument *fildes* is not valid, then a call to *fstat(fildes, buf)* returns a value of (*int*)-1 and sets *errno* to [EBADF].

5.6.3 *access*()

5.6.3.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function type `int access(const char *, int)` is declared. (See GA36 inproto 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *access*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *access*() is defined as a macro when the header <unistd.h> is included:

When the macro *access*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *access*() is defined as a macro in the header <unistd.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.6.3.2 Description

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *access*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(A) When the *path* argument points to the string "/", then *access*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

06(A) When the *path* argument points to the string "///", then *access*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *access*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *access*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *path* argument points to the string "F1/" and `F1` is a directory, then *access*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

10(A) When the argument *path* points to the string "F1//" and F1 is a directory, then *access*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *path* argument points to the string "F1/F2", then *access*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the *path* argument points to the string "F1/./F2", then *access*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string "F1/../F1/F2", then *access*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string "F1//F2", then *access*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:
When the *path* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *access*() resolves the pathname component by truncating it to {NAMEMAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *access(path, amode),* the pathname *path* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *access(path, amode),* the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) When the header `<unistd.h>` is included, the symbolic constants R_OK, W_OK, X_OK, and F_OK are defined.

19(A) When the file referenced by *path* exists, then a call to *access(path, amode)* passes tests for *amode* F_OK and returns a value of zero.

20(A) When the real user ID of the calling process matches the owner of the file referenced by *path* and permission (see Table 5.5) is granted for the owner, then a call to *access(path, amode)* succeeds for amode (see Table 5.5) and returns a value of zero.

Table 5.5—Access via File Owner Class

Permission	amode
Read	R_OK
Write	W_OK
Execute	X_OK

21(A) When the real user ID of the calling process does not match the owner of the file referenced by *path,* when the real group ID or one of the supplementary group IDs of the calling process matches the group owner of the

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

file, and when permission (see Table 5.6) is granted for the group, then a call to *access(path, amode)* succeeds for *amode* (see Table 5.6) and returns a value of zero.

Table 5.6—Access via File Group Class

Permission	amode
Read	R_OK
Write	W_OK
Execute	X_OK

22(A) When the real user ID of the calling process does not match the owner of the file referenced by *path*, when neither the real group ID nor any of the supplementary group IDs of the calling process match the group owner of the file, and when permission (see Table 5.7) is granted for others, then a call to *access(path, amode)* succeeds for amode (see Table 5.7) and returns a value of zero.

Table 5.7—Access via File Other Class

Permission	amode
Read	R_OK
Write	W_OK
Execute	X_OK

D01(C) If the implementation provides a mechanism to assign to a process the appropriate privileges to override the file access control mechanism:

The details are contained in 2.2.2.4 or 5.6.3.2 of the PCD.1.

D02(C) If the implementation provides a mechanism to assign to a process the appropriate privileges to override the file access control mechanism and it is documented whether the implementation supports a successful return from *access(path, X_OK)* for a process with appropriate privileges when none of the execute bits for *path* are set:

The details are contained in 5.6.3.2 of the PCD 1.

23(C) If the implementation provides a method for associating with a process the appropriate privilege to override the file access control mechanism:

When the process has the appropriate privileges to override the file access control mechanism and the file exists, then a call to *access(path, amode)* will succeed when *amode* is set to F_OK, R_OK, or W_OK.

Testing Requirements:

Test for *amode* set to F_OK, R_OK, and W_OK.

24(C) If the implementation provides a method for associating with a process the appropriate privilege to override the file access control mechanism:

When the process has the appropriate privileges to override the file access control mechanism, then a call to *access(path, amode)* will succeed when *amode* is set to X_OK and any of the execute bits are set or when the file is a directory.

25(A) When *amode* is a bitwise inclusive OR of two or more of R_OK, W_OK, and X_OK, and when separate calls with each of the indicated permissions requested independently would all return successfully, then a call to *access(path, amode)* returns a value of zero.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5.6.3.3 Returns

R01 When a call to *access*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 19–25 in 5.6.3.2.)

R02 When a call to *access*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertions 26–28 and 30–37 in 5.6.3.4.)

5.6.3.4 Errors

26(A) When any of the requested permissions specified by *amode* are denied, then a call to *access(path, amode)* returns a value of (*int*)-1 and sets *errno* to [EACCES].

27(A) When search permission is denied for a component of the path prefix of *path*, then a call to *access(path, amode)* returns a value of (*int*)-1 and sets *errno* to [EACCES].

28({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *access(path, amode)* returns a value of (*int*)-1 and sets *errno* to [EACCES].

29({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS_NAME_MAX}, then a call to *access(path, amode)* succeeds.

30(A) If {PATH_MAX} ≤ {PCTS_PATH_MAX}:

When the length of the *path* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *access(path, amode)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

Otherwise:

When the length of the path argument is {PCTS_PATH_MAX}, then a call to *access*() is successful.

31(A) When the *path* argument points to an empty string, then a call to *access(path, amode)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

32(A) When a component of the path prefix of *path* does not exist, then a call to *access(path, arnode)* returns a value of (*int*)-1 anti sets *errno* to [ENOENT].

33(A) When the *path* argument points to name of file that does not exist, then a call to *access(path, amode)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

34(A) When a component of the path prefix of *path* is not a directory, then a call to *access(path, amode)* returns a value of (*int*)-1 and sets *errno* to [ENOTDIR].

35(C) If the implementation supports a read-only file system:

When write access is requested for a file on a read-only file system, then a call to *access(path, amode)* returns a value, of (*int*)-1, sets *errno* to [EROFS], and does not mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the file. (See GA13 in 2.3.5.)

D03(C) If the implementation supports the detection of [EINVAL] for *access*():

The details under which [EINVAL] occurs for *access*() are contained in 5.6.3.4 of the PCD.1.

36(C) If the implementation supports the detection of [EINVAL] for *access*():

When the amode argument is invalid, then a call to *access(path, amode)* returns a value of (*int*)-1 and sets *errno* to [EINVAL].

37(D) If the implementation does not support the detection of [EINVAL] for *access*():

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A call to *access(path, amode)* is successful (unless a different error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 {4}.

5.6.4 *chmod*()

5.6.4.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <sys/stat.h> is included, then the function prototype `int chmod(const char *, mode_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <sys/stat.h> is included, then the function *chmod*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *chmod*() is defined as a macro when the header <sys/stat.h> is included:
When the macro *chmod*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *chmod*() is defined as a macro in the header <sys/stat.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.6.4.2 Description

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *chmod()* resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(PCTS APP_MODE && PCTS_ROOT_WRITABLE?A:UNTESTED)

When the *path* argument points to the string "/", then *chmod*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

06(PCTS APP_MODE && PCTS_ROOT_WRITABLE?A:UNTESTED)

When the *path* argument points to the string "/ / /". then *chmod*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *chmod*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *chmod*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *path* argument points to the string "F1/" and F1 is a directory, then chmod() resolves the pathname to F1, which is in the current working directory. (See GA19 in 2.3.6.)

10(A) When the argument *path* points to the string "F1//" and F1 is a directory, then *chmod*() resolves the pathname to F1, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *path* argument points to the string "F1/F2", then *chmod*() resolves the pathname to the file F2 in the directory F1, which is in the current working directory. (See GA21 in 2.3.6.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12(A) When the *path* argument points to the string "F1/./F2", then *chmod*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string "F1/../F1/F2", then *chmod*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string "F1//F2", then *chmod*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

When the *path* component is a string of more than {NAME_MAX} bytes in a directory for which LPOSIX_NO_TRUNC} is not supported,then *chmod*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *chmod(path, mode)*, the pathname *path* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *chmod(path, mode)*, the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) When search access is granted to each component of the path prefix of *path*, then the standard file access control mechanism does not cause *chmod*() to indicate that file access is denied.

19(A) When the effective user ID matches the user ID of the file, then a call to *chmod(path, mode)* sets the file permission bits of the file named by *path* according to the bit pattern contained in the argument *mode* and returns a value of zero.

D01(C) If the implementation provides a mechanism to assign to a process the appropriate privileges to change the file mode:

The details are contained in 2.2.2.4 or 5.6.4.2 of the PCD.1.

20(C) If the implementation provides the mechanism for creating a process with the appropriate privilege to change the file mode:

When the process has the appropriate privileges to change the file mode, then a call to *chmod(path, mode)* sets the file permission bits of the file named by *path* according to the bit pattern contained in the argument *mode* and returns a value of zero

21(PCTS_CHMOD_SET_IDS?A:UNTESTED)

When the effective user ID matches the user ID of the file, then a call to *chmod(path, mode)* sets S_ISUID, S_ISGID, and the file permission bits of the file named by *path* according to the bit pattern contained in the argument *mode* and returns a value of zero.

22(PCTS_CHMOD_SET_IDS?C:UNTESTED)

If the implementation provides a method for associating with a process the appropriate privileges to change the file mode:

When the process has the appropriate privileges to change the file mode, then a call to *chmod(path, mode)* sets S_ISUID, S_ISGID, and the file permission bits of the file named by *path* according to the bit pattern contained in the argument *mode* and returns a value of zero.

D02(C) If restrictions not specified by POSIX.1 {3} cause the S_ISUID or S_ISGID bits to be ignored:

The details when additional restrictions cause the S_ISUID and S_ISGID bits in mode to be ignored are contained in 5.6.4.2 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

23(PCTS_CHMOD_SET_IDS?A:UNTESTED)

When the calling process does not have the appropriate privileges to change the file mode, and when the group owner of the regular file specified by *path* does not match the effective group ID or one of the supplementary group IDs, then a call to *chmod(path, mode)* clears the S_ISGID bit upon successful completion.

D03(A) The effect on file descriptors for files open at the time of the *chmod*() function are contained in 5.6.4.2 of the PCD.1.

24(A) A successful call to *chmod(path, mode)* marks for update the *st_ctime* field of the file.

5.6.4.3 Returns

R01 When a call to *chmod*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 19–22 in 5.6.4.2.)

R02 When a call to *chmod(path, mode)* completes unsuccessfully, then a value of (*int*)-1 is returned, sets *errno* to indicate the error, and no change is made to the file *mode*. (See Assertions 25, 26, and 28–34 in 5.6.4.4.)

5.6.4.4 Errors

25(A) When search permission is denied for a component of the path prefix of *path*, then a call to *chmod(path, mode)* returns a value of (*int*)-1, sets *errno* to [EACCES], and does not change the file *mode*.

26({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with [_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *chmod(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

27({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS_NAME_MAX}, then a call to chmod(path, mode) succeeds.

28(A) If {PATH_MAX} > {PCTS_PATH_MAX}:

When the length of the *path* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *chmod(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

Otherwise:

When the length of the path argument is {PCTS_PATH_MAX}, then a call to *chmod*() is successful.

29(A) When a component of the *path* prefix of path is a not a directory, then a call to *chmod(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOTDIR].

30(A) When the file named by *path* does not exist, then a call to *chmod(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

31(A) When a component of the path prefix of *path* does not exist, then a call to *chmod(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

32(A) When the argument *path* points to an empty string, then a call to *chmod(path, mode)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

33(A) When the effective user ID does not match the owner of the file and the calling process does not have the appropriate privileges to change the file mode, then a call to *chmod(path, mode)* returns a value of (*int*)-1, sets *errno* to [EPERM], and does not change the file *mode*.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

34(C) If the implementation supports a read-only file system:
When the named file resides on a read-only file system, then a call to *chmod(path, mode)* returns a value of (*int*)-1, sets *errno* to [EROFS], does not change the mode of the file, and does not mark for update the *st_atime, st_ctime,* and *st_mtime* fields of the file. (See GA13 in 2.3.5.)

5.6.5 *chown*()

5.6.5.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype
`int chown(const char *, uid_t, gid_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included. then the function *chown*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *chown*() is defined as a macro when the header <unistd.h> is included:
When the macro *chown*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int.* (See GA37 in 2.7.3.)

03(C) If *chown*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.6.5.2 Description

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *chown*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(PCTS_APP_OWNER && PCTS_ROOT_WRITABLE?A:UNTESTED)

When the *path* argument points to the string "/", then *chown*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

06(PCTS_APP_OWNER && PCTS_ROOT_WRITABLE?A:UNTESTED)

When the *path* argument points to the string "/ / /", then *chown*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *chown*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *chown*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *path* argument points to the string "F1/" and `F1` is a directory, then *chown*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

10(A) When the argument *path* points to the string "F1//" and F1 is a directory, then *chown*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *path* argument points to the string "F1/F2", then *chown*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12(A) When the *path* argument points to the string "F1/./F2", then *chown*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string "F1/../F1/F2", then *chown*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string "F1//F2", then *chown*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

When the *path* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *chown*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *chown(path, owner, group)*, the pathname *path* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *chown(path, owner, group)*, the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) When search access is granted to each component of the path prefix of path, then the standard file access control mechanism does not cause *chown*() to indicate that file access is denied.

D01(C) If the implementation provides a mechanism to assign to a process the appropriate privileges to change file owner:

The details are confined in 2.2.2.4 or 5.6.5.2 of the PCD.1.

19(C) If the implementation provides a method for associating with a process the appropriate privilege to change the owner of a file:

When the process has the appropriate privileges to change the owner of a file, then a call to *chown(path, owner, group)* sets the user ID and group ID of the named file to the numeric values confined in *owner* and *group* respectively, and a value of zero is returned.

20(C) If the behavior associated with {_POSIX_CHOWN_RESTRICTED} is supported for the file:

When the effective user ID is equal to the user ID of the file, when the process does not have the appropriate privileges to change the owner of a file, when the argument owner is equal to the user ID of the file, and when the argument group is equal either to the effective group ID of the calling process or to one of its supplementary group IDs, then a call to *chown(path, owner, group)* sets the group ID of the file named by path to the numeric value contained in *group*, and a value of zero is returned.

21(C) If the behavior associated with {_POSIX_CHOWN RESTRICTED} is not supported for the file:

When the effective user ID matches the user ID of the current file, then a call to *chown(path, owner, group)* sets the user ID and group ID of the file named by *path* to the numeric values contained in *owner* and *group* respectively, and a value of zero is returned.

D02(C) If the implementation provides a mechanism to assign to a process the appropriate privileges to change file owner:

The effect on the S_ISUID and S_ISGID bits of the file mode of a regular file upon successful return from *chown*() from a process with appropriate privileges are contained in 2.2.2.4 or 5.6.5.2 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22(PCTS_CHMOD_SET_IDS?A:UNTESTED)

When the path argument refers to a regular file, and when the call is made by a process without the appropriate privileges to change the owner of a file, then a successful call to *chown(path, owner, group)* clears the set-user-ID and set-group-ID bits of the file mode.

23(A) A successful call to *chown*() marks for update the *st_crime* time-related field of the file.

5.6.5.3 Returns

R01 When a call to *chown*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 19–21 in 5.6.5.2.)

R02 When a call to *chown*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error, and no change is made to the owner and group of the file. (See Assertions 24, 25, and 27–37 in 5.6.5.4.)

5.6.5.4 Errors

24(A) When search permission is denied for a component of the path prefix of *path*, then a call to *chown(path, owner, group)* returns a value of (*int*)-1, sets *errno* to [EACCES], and does not change the owner and group of the file.

25({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *chown(path, owner, group)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

26({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS_NAME_MAX}, then a call to *chown(path, owner, group)* succeeds.

27(A) If {PATH_MAX} ≤ {PCTS_PATH_MAX}:

When the length of the *path* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *chown(path, owner, group)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

Otherwise:

When the length of the path argument is {PCTS_PATH_MAX}, then a call to *chown*() is successful.

28(A) When a component of the path prefix of *path* is not a directory, then a call to *chown(path, owner, group)* returns a value of (*int*)-1 and sets *errno* to [ENOTDIR].

29(A) When the file named by *path* does not exist, then a call to *chown(path, owner, group)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

30(A) When a component of the path prefix of *path* does not exist, then a call to *chown(path, owner, group)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

31(A) When the *path* argument points to an empty string, then a call to *chown(path, owner, group)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

32(C) If the behavior associated with {_POSIX_CHOWN_RESTRICTED} is supported for the file

When the calling process does not have the appropriate privileges to change the owner of a file and the owner does not match the user ID of the file, then a call to *chown(path, owner, group)* returns a value of (*int*)-1, sets *errno* to [EPERM], and does not change the owner and group of the file.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

33(C) If the behavior associated with {_POSIX_CHOWN_RESTRICTED} is supported for the file:

When the calling process does not have the appropriate privileges to change the owner of a file, and when the group does not match either the effective group ID of the process or any supplementary group ID, then a call to *chown(path, owner, group)* returns a value of (*int*)-1, sets *errno* to [EPERM], and does not change the *owner* and *group* of the file.

34(A) When the argument owner does not match the effective user ID, then a call to *chown(path, owner, group)* returns a value of (*int*)-1, sets *errno* to [EPERM], and does not change the owner and group of the file.

35(C) If the implementation supports a read-only file system:

When the named file resides on a read-only file system, then a call to *chown(path, owner, group)* returns a value of (*int*)-1, sets *errno* to [EROFS], does not change the owner and group of the file, and does not mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the file. (See GA13 in 2.3.5.)

D03(C) If the implementation supports the detection of [EINVAL] for *chown*():

The details under which [EINVAL] occurs for *chown*() are contained in 5.6.5.4 of the PCD.1. (See DGA02 in 2.4.)

36(PCTS_INVALID_OWNER?C:UNTESTED)

If the implementation provides the mechanism for creating a process with the appropriate privilege to change the owner and group ID of a file and the implementation supports the detection of [EINVAL] for *chown*():

When the calling process has the appropriate privileges to change the owner and group ID, and when either the owner or group ID supplied is invalid, then a call to *chown(path, owner, group)* returns a value of (*int*)-1, sets *errno* to [EINVAL], and does not change the owner and group of the file.

37(D) If the implementation provides the mechanism for creating a process with the appropriate privilege to change the owner and group ID of a file and the implementation does not support the detection of [EINVAL] for *chown*():

When the calling process has the appropriate privileges to change the owner or group ID, and when either the owner or group ID supplied is invalid, then a call to *chown(path, owner, group)* is successful (unless a different error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 {4}.

5.6.6 *utime*()

5.6.6.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <utime.h> is included, then the function prototype `int utime(const char *, const struct utimbuf *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <utime.h> is included, then the function *utime*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *utime*() is defined as a macro when the header <utime.h> is included:

When the macro *utime*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type int. (See GA37 in 2.7.3.)

03(C) If *utime*() is defined as a macro in the header <utime.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860303 507 ■

5.6.6.2 Description

04(A) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *utime*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(PCTS_APP_TIMES && PCTS_ROOT_WRITABLE?A:UNTESTED)

When the *path* argument points to the string "/", then *utime*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

06(PCTS_APP_TIMES && PCTS_ROOT_WRITABLE?A:UNTESTED)

When the *path* argument points to the string "///", then *utime*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

07(A) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *utime*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *utime*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *path* argument points to the string "F1/" and `F1` is a directory, then *utime*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

10(A) When the argument *path* points to the string "F1//" and F1 is a directory, then *utime*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *path* argument points to the string "F1/F2", then *utime*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the *path* argument points to the string "F1/./F2", then *utime*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *path* argument points to the string "F1/../F1/F2", then *utime*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *path* argument points to the string "F1/F2" then *utime*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

When the *path* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX NO_TRUNC} is not supported, then *utime*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *utime*(*path*, *times*), the pathname *path* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *utime*(*path*, *times*), the pathname *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) When *utime*(*path*, *times*) is granted write access to *path* and search access to the path prefix of *path*, then the standard file access control mechanism does not cause *utime*() to indicate that file access is denied.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

19(A) When the *times* argument is NULL and the effective user ID matches the owner of the file, then a call to *utime*(*path*, *times*) sets the last accessed time and last modified time to the current time and returns a value of zero.

20(A) When the times argument is NULL and the process has write permission on the file, then a call to *utime*(*path*, *times*) sets the last accessed time and last modified time to the current time and returns a value of zero.

21(A) When the *times* argument is not NULL and addresses a valid *utimbuf* structure, and when the effective user ID matches the owner of the file, then a call to *utime*(*path*, *times*) sets the last accessed time and last modified time to the *actime* and *modtime* fields, respectively, of the structure *utimbuf* and returns a value of zero.

Testing Requirements:

When the *utimbuf* structure contains additional members beyond those specified in POSIX.1 {3}, and when the effect of these members is not activated, then these additional structure members do not affect the behavior of the call to *utime*().

22(C) If the implementation provides the mechanism for creating a process with the appropriate privilege to change the file times:

When the process has the appropriate privileges to change the file times, and when the effective user ID does not match the owner of the file, then a call to *utime*(*path*, *times*) sets the last accessed time and last modified time to the *actime* and *modtime* fields, respectively, of the structure *utimbuf* addressed by the non-NULL argument *times* and returns a value of zero

Testing Requirements:

When the *utimbuf* structure contains additional members beyond those specified in POSIX.1 {3}, and when the effect of these members is not activated, then these additional structure members do not affect the behavior of the call to *utime*().

23(C) If the implementation provides the mechanism for creating a process with the appropriate privilege to change the file times:

When the process has the appropriate privileges to change the file times, then a call to *utime*(*path*, *NULL*) sets the last accessed time and last modified time to the current time and returns the value zero.

24(A) When the header `<utime.h>` is included, then the elements *asctime* and *modtime* are defined in struct *utimbuf* as type *time_t*.

D01(C) If the *utimbuf* structure includes members other than those specified by POSIX.1 {3}, and this is documented:

The details are contained in 5.6.6.2 of the PCD.1.

25(A) A call to *utime*() marks for update the *st_ctime* time-related field of the file.

5.6.6.3 Returns

R01 When a call to *utime*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 19–23 in 5.6.6.2.)

R02 When a call to *utime*() completes unsuccessfully, then a value of (*int*)−1 is returned, sets *errno* to indicate the error, and the file times are not affected. (See Assertions 26–28 and 30–36 in 5.6.6.4.)

5.6.6.4 Errors

26(A) When search permission is denied for a component of the path prefix of *path*, then a call to *utime*(*path*, *times*) returns a value of (*int*)-1, sets *errno* to [EACCES], and does not affect the file times.

27(A) When the times argument is NULL, when the effective user ID of the process does not match the owner of the file, and when write access is denied, then a call to *utime*(*path*, *times*) returns a value of (*int*)-1, sets *errno* to [EACCES], and does not affect the file times.

28({NAME_MAX}_≤{PCTS_NAME_MAX}?C:UNTESTED)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of path is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *utime(path, times)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

29({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS_NAME_MAX}, then a call to *utime(path, times)* succeeds.

30(A) If {PATH_MAX} ≤ {PCTS_PATH_MAX}:

When the length of the *path* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *utime(path, times)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

Otherwise:

When the length of the path argument is {PCTS_PATH_MAX}, then a call to *utime*() is successful.

31(A) When the file named by *path* does not exist, then a call to *utime(path, times)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

32(A) When a component of the path prefix of *path* does not exist, then a call to *utime(path, times)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

33(A) When the *path* argument points to an empty string, then a call to *utime(path, times)* returns a value of (*int*)-1 and sets *errno* to [ENOENT].

34(A) When a component of the path prefix of *path* is not a directory, then a call to *utime(path, times)* returns a value of (*int*)-1 and sets *errno* to [ENOTDIR].

35(A) When the *times* argument is not NULL, when the effective user ID of the calling process has write access to the file but does not match the owner of the file, and when the calling process does not have the appropriate privileges to change the file times, then a call to *utime(path, times)* returns a value of (*int*)-1, sets *errno* to [EPERM], and does not affect the file times.

36(C) If the implementation supports a read-only file system:

When the named file resides on a read-only file system, then a call to *utime(path, times)* returns a value of (*int*)-1, sets *errno* to [EROFS], and does not mark for update the *st_atime*, *st_ctime*, and *st_mtime* fields of the file. (See GA13 in 2.3.5.)

5.7 Configurable Pathname Variables

5.7.1 Get Configurable Pathname Variables

5.7.1.1 *pathconf*() (5.7.1)

5.7.1.1.1 Synopsis (5.7.1.1)

01(A) If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `long pathconf(const char *, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *pathconf*() is declared with the result type *long*. (See GA36 in 2.7.3.)

02(C) If *pathconf*() is defined as a macro when the header <unistd.h> is included:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the macro *pathconf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *long*. (See GA37 in 2.7.3.)

03(C) If *pathconf*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

5.7.1.1.2 Description (5.7.1.2)

04(B) When the first filename component of the *path* argument is ".", and the pathname does not begin with a slash, then *pathconf*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

05(B) When the path argument points to the string "/" then *pathconf*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

06(B) When the *path* argument points to the string "///", then *pathconf*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

07(B) When the *path* argument points to a string beginning with a single slash or beginning with three or more slashes, then *pathconf*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

08(B) When the first filename component of the *path* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *pathconf*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

09(B) When the *path* argument points to the string "F1/" and `F1` is a directory, then *pathconf*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

10(B) When the argument *path* points to the string "F1//" and `F1` is a directory, then *pathconf*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

11(B) When the *path* argument points to the string "F1/F2", then *pathconf*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

12(B) When the *path* argument points to the string "F1//F2", then *pathconf*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

13(B) When the *path* argument points to the string "F1/../F1/F2", then *pathconf*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

14(B) When the *path* argument points to the string "F1//F2", then *pathconf*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 [4].

15(D) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

When the *path* component is a string of more, than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *pathconf*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

See Reason 3 in Section 5. of POSIX.3 [4].

16(A) On a call to *pathconf(path, name)* the pathname *path* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 5.1.1.)

17(B) On a call to *pathconf(path, name)*, the pathnome *path* retains the unique identity of its upper- and lowercase letters. (See GA03 in 5.1.1.)

See Reason 3 in Section 5. of POSIX.3 [4].

18(A) When *pathconf(path, name)* is granted search access to the path prefix of *path*, then the standard file access control mechanism does not cause *pathconf*() to indicate that file access is denied.

19(A) When the header `<unistd.>` is included, then the symbolic constants _PC_LINK_MAX, _PC_MAX_CANON, _PC_MAX_INPUT, _PC_NAME_MAX,_PC_PATH_MAX, _PC_PIPE_BUF, _PC_CHOWN_RESTRICTED, _PC_NO_TRUNC, and _PC_VDISABLE are defined and are distinct.

20(A) A call to *pathconf(path, _PC_LINK_MAX)* either returns a value greater than or equal to {_POSIX_LINK_MAX} or returns -1.

21(C) If LINK_MAX is defined when `<limits.h>` is included:

A call to *pathconf(path, _PC_LINK_MAX)* returns a value greater than or equal to the value defined for LINK_MAX in `<limits.h>`.

22(PCTS_GTI_DEVICE?A:UNTESTED)

When *path* refers to a terminal file, then a call to *pathconf(path, _PC_MAX_CANON)* either returns a value greater than or equal to {_POSIX_MAX_CANON} or returns −1.

23(PCTS_GTI_DEVICE?C:UNTESTED)

If MAX_CANON is defined when `<limits.h>` is included:

When *path* refers to a terminal file, then a call to *pathconf(path, _PC_MAX_CANON)* returns a value greater than or equal to the value defined for MAX_CANON in `<limits.h>`.

24(PCTS_GTI_DEVICE?A:UNTESTED)

When *path* refers to a terminal file, then a call to *pathconf(path, _PC_MAX_INPUT)* either returns a value greater than or equal to {_POSIX_MAX_INPUT} or returns -1.

25(PCTS_GTI_DEVICE?C:UNTESTED)

If MAX_INPUT is defined when `<limits.h>` is included:

When *path* refers to a terminal file, then a call to *pathconf(path, _PC_MAX_INPUT)* returns a value greater than or equal to the value defined for MAX_INPUT in `<limits.h>`.

26(A) When *path* refers to a directory, then a call to *pathconf(path, _PC_NAME_MAX)* either returns a value greater than or equal to {_POSIX_NAME_MAX} or returns −1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

27(C) If NAME_MAX is defined when <limits.h> is included:
When *path* refers to a directory, then a call to *pathconf(path, _PC_NAME_MAX)* returns a value greater than or equal to the value defined for NAME_MAX in <limits.h>.

28(A) When *path* refers to a directory, then the value returned by *pathconf(path, _PC_NAME_MAX)* applies to the filenames within the directory.

29(A) When *path* refers to a directory, then a call to *pathconf(path, PC_PATH_MAX)* either returns a value greater than or equal to {_POSIX_PATH_MAX} or returns -1.

30(C) If PATH MAX is defined when <limits.h> is included:
When *path* refers to a directory, then a call to *pathconf(path, PC_PATH_MAX)* returns a value greater than or equal to the value defined for PATH_MAX in <limits.h>.

31(A) When *path* refers to the current working directory, then the value returned by *pathconf(path, _PC_PATH_MAX)* is the maximum length of a relative pathname—the terminating null character need not be included in the length.

32(A) When *path* refers to a FIFO or directory, then a call to *pathconf(path, _PC_PIPE_BUF)* either returns a value greater than or equal to {_POSIX_PIPE_BUF} or returns -1.

33(C) If PIPE_BUF is defined when <limits.h> is included:
When *path* refers to a FIFO or directory, then a call to *pathconf(path, _PC_PIPE_BUF)* returns a value greater than or equal to the value defined for PIPE_BUF in <limits.h>.

34(A) When *path* refers to a FIFO, then the value returned by *pathconf(path, _PC_PIPE_BUF)* applies to the FIFO.

35(A) When *path* refers to a directory, then the value returned by *pathconf(path, _PC_PIPE_BUF)* applies to any FIFO that exists or can be created within the directory.

36(C) If _POSIX_CHOWN_RESTRICTED is defined when <unistd.h> is included:
A call to *pathconf(path, PC_CHOWN_RESTRICTED)* returns a value = {_POSIX_CHOWN_RESTRICTED}.

37(A) When *path* refers to a directory, then the value returned by *pathconf(path, PC_CHOWN_RESTRICTED)* applies to any files, other than directories, that exist or can be created within the directory.

38(C) If_POSIX_NO_TRUNC is defined when <unistd.h> is included:
When *path* refers to a directory, then a call to *pathconf(path, _PC_NO_TRUNC)* returns a value = {_POSIX_NO_TRUNC}.

39(A) When *path* refers to a directory, then the value returned by *pathconf(path, _PC_NO_TRUNC)* applies to the filenames within the directory.

40(PCTS_GTI_DEVICE?C:UNTESTED)

If_POSIX_VDISABLE is defined when <unistd.h> is included:
When *path* refers to a terminal file, then a call to *pathconf(path, _PC_VDISABLE)* returns the same value as that defined for _POSIX_VDISABLE in <unistd.h>.

41(PCTS_GTI_DEVICE?A:UNTESTED)

When *path* refers to a terminal, then the value returned by *pathconf(path, _PC_VDISABLE)* can be used to disable the effects associated with the receipt of a special control character from the terminal.

D01(C) If the implementation supports additional configurable pathname variables beyond those listed in Table 5.2 of POSIX.1 [3], and this is documented:
The details are contained in 5.7.1.2 of the PCD.1.

D02(C) If the implementation supports the association of the variable names {MAX_CANON}, {MAX_INPUT}, or {_POSIX_VDISABLE} in a call to *pathconf()* with other than a terminal file, and this is documented:
The details are contained in 5.7.1.2 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D03(C) If the implementation supports the association of the variable names {NAME_MAX}, {PATH_MAX}, or {_POSIX_NO_TRUNC} in a call to *pathconf*() with file types other than a directory, and this is documented:
The details are contained in 5.7.1.2 of the PCD.1.

D04(C) If the implementation supports the association of the variable name {PIPE_BUF} in a call to *pathconf*() with file types other than a pipe, a FIFO, or a directory, and this is documented:
The details are contained in 5.7.1.2 of the PCD.1.

5.7.1.1.3 Returns (5.7.1.3)

R01 When a call to *pathconf*() completes successfully, then the current pathable value for the file or directory is returned without changing *errno*. (See Assertions 4–41 in 5.7.1.1.2.)

R02 When a call to *pathconf*() completes unsuccessfully, then a value of (*long*)–1 is returned and sets *errno* to indicate the error. (See Assertions 42–51 and 53–57 in 5.7.1.1.4.)

5.7.1.1.4 Errors (5.7.1.4)

D05(C) If the implementation supports the detection of [EINVAL] for *pathconf*():
The details under which [EINVAL] occurs for *pathconf*() are contained in 5.7.1.4 of the PCD.1. (See DGA02 in 2.4.)

42(A) When the value of *name* is invalid, then a call to *pathconf(path, name)* returns a value of (*long*)-1 and sets *errno* to [EINVAL].

D06(C) If the implementation supports the detection of [EACCES] for *pathconf*():
The details under which [EACCES] occurs for *pathconf*() are contained in 5.7.1.4 of the PCD.1. (See DGA02 in 2.4.)

43(A) When search permission is denied for a component of the path prefix of *path*, then a call to *pathconf(path, name*) either returns a value of ((*long*)-1) and sets *errno* to [EACCES] or returns the value associated with *name* for file named *path*.

44(A) When *path* does not refer to a terminal file, then a call to *pathconf(path, _PC_MAX_CANON)* either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {MAX_CANON}.

45(A) When *path* does not refer to a terminal file, then a call to *pathconf(path, _PC_MAX_INPUT)* either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {MAX_INPUT}.

46(A) When *path* does not refer to a directory, then a call to *pathconf(path, _PC_NAME_MAX)* either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {NAME_MAX}.

47(A) When *path* does not refer to a directory, then a call to *pathconf(path, _PC_PATH_MAX)* either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {PATH_MAX}.

48(A) When *path* does not refer to a FIFO or directory, then a call to *pathconf(path, _PC PIPEBUF)* either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {PIPE_BUF}.

49(A) When *path* does not refer to a directory, then a call to *pathconf(path, _PC_NO_TRUNC)* either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {_POSIX_NO_TRUNC}.

50(A) When *path* does not refer to a terminal file, then a call to *pathconf(path, _PC_VDISABLE)* either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {_POSIX_VDISABLE}.

D07(C) If the implementation supports the detection of [ENAMETOOLONG] for *pathconf*():
The details under which [ENAMETOOLONG] occurs for *pathconf*() are contained in 5.7.1.4 of the PCD.1. (See DGA02 in 2.4.)

51({NAME_MAX}≤{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX NO TRUNC} is supported for the file:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the length of a pathname component of *path* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *pathconf(path, name)* either returns a value of ((*long*)-1) and sets *errno* to [ENAMETOOLONG] or returns a value appropriate for *name.*

52({NAME_MAX}>{PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the file:

When the length of a pathname component of *path* equals {PCTS_NAME_MAX}, then a call to *pathconf(path, name)* succeeds.

53(A) If {PATH_MAX} ≤ {PCTS_PATH_MAX}:

When the length of the *path* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *pathconf(path, name)* either returns a value of ((*long*)-1) and sets *errno* to [ENAMETOOLONG] or returns a value appropriate for *name.*

Otherwise:

When the length of the path argument is {PCTS_PATH_MAX}, then a call to *pathconf*() is successful.

D08(C) If the implementation supports the detection of [ENOENT] for *pathconf*():

The details under which [ENOENT] occurs for *closedir*() are contained in 5.7.1.4 of the PCD.1. (See DGA02 in 2.4.)

54(A) When the file named by *path* does not exist, then a call to *pathconf(path, name)* either returns a value of ((*long*)-1) and sets *errno* to [ENOENT] or returns a value appropriate for *name.*

55(A) When a component of the path prefix of *path* does not exist, then a call to *pathconf(path, name)* either returns a value of ((*long*)-1) and sets *errno* to [ENOENT] or returns a value appropriate for *name.*

56(A) When the *path* argument points to an empty string. then a call to *pathconf(path, name)* either returns a value of ((*long*)-1) and sets *errno* to [ENOENT] or returns a value appropriate for *name.*

D09(C) If the implementation supports the detection of [ENOTDIR] for *pathconf*():

The details under which this error occurs are contained in 5.7.1.4 of the PCD.1. (See DGA02 in 2.4.)

57(A) When a component of the path prefix of *path* is not a directory, then a call to *pathconf(path, name)* either returns a value of ((*long*)-1) and sets *errno* to [ENOTDIR] or returns a value appropriate for *name.*

5.7.1.2 *fpathconf*() (5.7.1)

5.7.1.2.1 Synopsis (5.7.1.1)

01(A) If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `long fpathconf(int, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *fpathconf*() is declared with the result type *long.* (See GA36 in 2.7.3.)

02(C) If *fpathconf*() is defined as a macro when the header <unistd.h> is included:

When the macro *fpathconf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *long.* (See GA37 in 2.7.3.)

03(C) If *fpathconf*() is defined as a macro in the header <unistd.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5.7.1.2.2 Description (5.7.1.2)

04(A) A call to *fpathconf(fildes, _PC_LINK_MAX)* returns a value greater than or equal to {_POSIX_LINK_MAX}.

05(C) If LINK_MAX is defined when <limits.h> is included:
A call to *fpathconf(fildes, _PCLINK_MAX)* returns a value greater than or equal to the value defined for LINK_MAX in <limits.h>.

06(PCTS_GTI_DEVICE?A:UNTESTED)

When *fildes* refers to a terminal file, then a call to *fpathconf(fildes, _PC_MAX_CANON)* either returns a value greater than or equal to {_POSIX_MAX_CANON} or returns −1.

07(PCTS_GTI_DEVICE?C:UNTESTED)

If MAX_CANON is defined when <limits.h> is included:
When *fildes* refers to a terminal file, then a call to *fpathconf(fildes, PCMAX_CANON)* returns a value greater than or equal to the value defined for MAX_CANON in <limits.h>.

08(PCTS_GTI_DEVICE?A:UNTESTED)

When *fildes* refers to a terminal file, then a call to *fpathconf(fildes, _PC_MAX_INPUT)* either returns a value greater than or equal to {_POSIX_MAX_INPUT} or returns −1.

09(PCTS_GTI_DEVICE?C:UNTESTED)

If {MAX_INPUT} is defined when <limits.h> is included:
When *fildes* refers to a terminal file, then a call to *fpathconf(fildes, _PC_MAX_INPUT)* returns a value greater than or equal to the value defined for MAX_INPUT in <limits.h>.

10(A) When *fildes* refers to a directory, then a call to *fpathconf(fildes, _PC_NAME_MAX)* either returns a value greater than or equal to {_POSIX_NAME_MAX} or returns −1.

11(C) If NAME_MAX is defined when <limits.h> is included:
When *fildes* refers to a directory, then a call to *fpathconf(fildes, _PC_NAME_MAX)* returns a value greater than or equal to the value defined for NAME_MAX in <limits.h>.

12(A) When *fildes* refers to a directory, then the value returned by *fpathconf(fildes, _PC_NAME_MAX)* applies to the filenames within the directory.

13(A) When *fildes* refers to a directory, then a call to *fpathconf(fildess, _PC_PATH_MAX)* either returns a value greater than or equal to {_POSIX_PATH_MAX} or returns −1.

14(C) If PATH_MAX is defined when <limits.h> is included:
When *fildes* refers to a directory, then a call to *fpathconf(fildes, _PC_PATH_MAX)* returns a value greater than or equal to the value defined for PATH_MAX in <limits.h>.

15(A) When *fildes* refers to the current working directory, then the value returned by *fpathconf(fildes, _PC_PATH_MAX)* is the maximum length of a relative pathname—the terminating null character need not be included in the length.

16(A) When *fildes* refers to a pipe, a FIFO, or a directory, then a call to *fpathconf(fildes, _PC_PIPE_BUF)* either returns a value greater than or equal to {_POSIX_PIPE_BUF} or returns -1.

Testing Requirements:
Test for file types of pipe, FIFO, and directory.

17(C) If PIPE_BUF is defined when <limits.h> is included:
When fildes refers to a pipe, a FIFO, or a directory, then a call to *fpathconf(fildes, _PC_PIPE_BUF)* returns a value greater than or equal to the value defined for PIPE_BUF in <limits.h>.
Testing Requirements:
Test for file types of pipe, FIFO, and directory.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

18(A) When *fildes* refers to a pipe or a FIFO, then the value returned by *fpathconf(fildes, _PC_PIPE_BUF)* applies to the pipe or the FIFO.

Testing Requirements:

Test for file types of pipe and FIFO.

19(A) When *fildes* refers to a directory, then the value returned by *fpatheonf(fildes, _PC_PIPE_BUF)* applies to any FIFOs that exist or can be created within the directory.

20(C) If _POSIX_CHOWN_RESTRICTED is defined when <unistd.h> is included:

A call to *fpathconf(fildes, _PC_CHOWN_RESTRICTED)* returns a value equal to {_POSIX_CHOWN_RESTRICTED}.

21(A) When *fildes* refers to a directory, then the value returned by *fpathconf(fildes, _PC_CHOWN_RESTRICTED)* applies to all files other than directories that exist or can be created within the directory.

22(C) If {_POSIX_NO_TRUNC} is defined when <unistd.h> is included:

When *fildes* refers to a directory, then a call to *fpathconf(fildes, _PC_NO_TRUNC)* returns a value equal to {_POSIX_NO_TRUNC}.

23(A) When *fildes* refers to a directory, then the value returned by *fpathconfi(fildes, _PC_NO_TRUNC)* applies to the filenames within the directory.

24(PCTS_GTI_DEVICE?C:UNTESTED)

If {_POSIX_VDISABLE} is defined when <unistd.h> is included:

When fildes refers to a terminal file, then a call to *fpathconf(fildes, PC_VDISABLE)* returns the same value as that defined for _POSIX_VDISABLE in <unistd.h>.

25(PCTS_GTI_DEVICE?A:UNTESTED)

When *fildes* refers to a terminal, then the value returned by *fpathconf(fildes, _PC_VDISABLE)* can be used to disable the effects associated with the receipt of a special control character from the terminal.

D01(C) If the implementation supports the association of the variable names {MAX_CANON}, {MAX_INPUT}, or {_POSIX_VDISABLE} in a call to fpathconf() with a file type other than a terminal file, and this is documented:

The details are contained in 5.7.1.2 of the PCD.1.

D02(C) If the implementation supports the association of the variable names {NAME_MAX}, {PATH_MAX}, or {_POSIX_NO_TRUNC} in a call to fpathconf() with a file type other than a directory, and this is documented:

The details are contained in 5.7.1.2 of the PCD.1.

D03(C) If the implementation supports the association of the variable name {PIPE_BUF} in a call to *fpathconf()* with other than a pipe, a FIFO, or a directory, and this is documented:

The details are contained in 5.7.1.2 of the PCD.1.

5.7.1.2.3 Returns (5.7.1.3)

R01 When a call to *fpathconf()* completes successfully, then the current variable value for the file or directory is returned wit, hour changing *errno*. (See Assertions 4–25 in 5.7.1.2.2.)

R02 When a call to *fpathconf()* completes unsuccessfully, then a value of (*long*)–1 is returned and sets *errno* to indicate the error. (See Assertions 26–34 in 5.7.1.2.4.)

5.7.1.2.4 Errors (5.7.1.4)

26(A) When the value of *name* is invalid, then a call to *fpathconf(fildes, name)* returns a value of (*long*)-1 and sets *errno* to [EINVAL].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D04(C) If the implementation supports the detection of [EBADF] for *fpathconf*():
The details under which this error occurs are contained in 5.7.1.4 of the PCD.1. (See DGA02 in 2.4.)

27(A) When the *fildes* argument is not a valid file descriptor, then a call to *fpathconf(fildes, name)* either returns a value of ((*long*)-1) and sets *errno* to [EBADF] or returns a value associated with *name.*

D05(C) If the implementation supports the detection of [EINVAL] for *fpathconf*():
The details under which this error occurs are contained in 5.7.1.4 of the PCD.1. (See DGA02 in 2.4.)

28(A) When *fildes* does not refer to a terminal file, then a call to *fpathconf(fildes, _PC_MAX_CANON)* either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {MAX_CANON}.

29(A) When *fildes* does not refer to a terminal file, then a call to*fpathconf(fildes, _PC_MAX_INPUT)* either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {MAX_INPUT}.

30(A) When *fildes* does not refer to a directory, then a call to *fpathconf(fildes, _PC_NAME_MAX)* either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {NAME_MAX}.

31(A) When *fildes* does not refer to a directory, then a call to *fpathconf(fildes, _PC_PATH_MAX)* either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {PATH_MAX}.

32(A) When *fildes* does not refer to a pipe, a FIFO, or a directory, then a call to *fpathconf(fildes, _PC_PIPE_BUF)* either returns a value of ((*long*)–1) and sets *errno* to [EINVAL] or returns a valid value for {PIPE_BUF}.

33(A) When *fildes* does not refer to a directory, then a call to *fpathconf(fildes, _PC_NO_TRUNC*) either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {_POSIX_NO_TRUNC}.

34(A) When *fildes* does not refer to a terminal file, then a call to *fpathconf(fildes, _PC_VDISABLE*) either returns a value of ((*long*)-1) and sets *errno* to [EINVAL] or returns a valid value for {_POSIX_VDISABLE}.

6. Input and Output Primitives

6.1 Pipes

6.1.1 *pipe*()

6.1.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `int pipe(int [2])` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *pipe*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *pipe*() is defined as a macro when the header <unistd.h> is included:
When the macro *pipe*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int.* (See GA37 in 2.7.3.)

03(C) If *pipe*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.1.1.2 Description

04(A) A successful call to *pipe(fildes)* creates a pipe, places two file descriptors in the array *fildes*, and returns a value of zero.

05(A) When a call to *pipe*() completes successfully, then data can be written to *fildes[1]*.

06(A) When a call to *pipe*() completes successfully, then data can be read from *fildes[0]*.

07(A) The two file descriptors placed in the array *fildes* by a call to *pipe(fildes)* are the two lowest file descriptors available at the time of the *pipe*() call.

08(A) When a call to *pipe*() completes successfully, then the O_NONBLOCK flag is cleared on both file descriptors.

09(A) When a call to pipe completes successfully, then the FD_CLOEXEC flag is clear on both file descriptors.

10(A) A call to *read*() on *fildes[0]* accesses the data written to *fildes[1]* on a first-in-first-out basis.

11(A) A call to *pipe*() marks for update the *st_atime*, *st_ctime*, and *st_mtime* time-related fields of the pipe.

6.1.1.3 Returns

R01 When a call to *pipe*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 4 in 6.1.1.2.)

R02 When a call to *pipe*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertions 12 and 13 in 6.1.1.4.)

6.1.1.4 Errors

12(A) If {OPEN_MAX} ≤ {PCTS_OPEN_MAX}:

When {OPEN_MAX} -1 or more files are opened, then a call to *pipe(fildes)* returns a value of (*int*)-1 and sets *errno* to [EMFILE].

Otherwise:

{PCTS_OPEN_MAX} files can be opened.

13(B) When the number of open files in the system would exceed a system-imposed limit, then a call to *pipe(fildes)* returns a value of (*int*)-1 and sets *errno* to [ENFILE].

See Reason 1 in Section 5. of POSIX.3 {4}.

6.2 File Descriptor Manipulation

6.2.1 Duplicate an Open File Descriptor

6.2.1.1 *dup*() (6.2.1)

6.2.1.1.1 Synopsis (6.2.1.1)

01(A) If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `int dup(int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *dup*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *dup*() is defined as a macro when the header <unistd.h> is included:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the macro *dup*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *dup*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

6.2.1.1.2 Description (6.2.1.2)

04(A) A call to *dup(fildes)* returns a new file descriptor with the following attributes:
- It is the lowest numbered available file descriptor.
- It refers to the same open file description (file pointer, access mode, file status flags) as the original file descriptor.
- It shares the same locks as the original file descriptor.
- It has the FD_CLOEXEC flag clear.

6.2.1.1.3 Returns (6.2.1.3)

R01 When a call to *dup*() completes successfully, then the lowest numbered available file descriptor is returned. (See Assertion 4 in 6.2.1.1.2.)

R02 When a call to *dup*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error (See Assertions 5 and 6 in 6.2.1.1.4.)

6.2.1.1.4 Errors (6.2.1.4)

05(A) When *fildes* is not a valid open file descriptor, then a call to *dup(fildes)* returns a value of (*int*)-1 and sets *errno* to [EBADF].

06(A) If {OPEN_MAX} ≤ {PCTS_OPEN_MAX}:
When {OPEN_MAX} files are open, then a call to *dup(fildes)* returns a value of (*int*)-1 and sets *errno* to [EMFILE].

Otherwise:
{PCTS_OPEN_MAX} files can be opened.

6.2.1.2 *dup2*() (6.2.1)

6.2.1.2.1 Synopsis (6.2.1.1)

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `int dup2(int, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *dup2*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *dup2*() is defined as a macro when the header <unistd.h> is included:
When the macro *dup2*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *dup2*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.2.1.2.2 Description (6.2.1.2)

04(A) When *fildes* is a valid file descriptor and is equal to *fildes2*, then a call to *dup2(fildes, fildes2)* returns *fildes2* without closing it.

05(A) A call to *dup2(fildes, fildes2)* returns a new file descriptor with the following attributes:

— Its value is equal to *fildes2*.
— It refers to the same open file description (file pointer, access mode, file status flags) as *fildes*.
— It shares the same locks as *fildes*.
— It has the FD_CLOEXEC flag clear.

6.2.1.2.3 Returns (6.2.1.3)

R01 When a call to *dup2*() completes successfully, then *fildes2* is returned. (See Assertion 4 in 6.2.1.2.2.)

R02 When a call to *dup2(fildes, fildes2)* completes unsuccessfully, then a value of (*int*)-1 is returned, sets *errno* to indicate the error, and the file referenced by *fildes2* is not closed. (See Assertion 6 in 6.2.1.2.4.)

6.2.1.2.4 Errors (6.2.1.4)

06(A) When *fildes* is not a valid open file descriptor, then a call to *dup2(fildes, fildes2)* returns a value of (*int*)-1, sets *errno* to [EBADF], and does not close argument *fildes2*.

07(A) When *fildes2* is negative, then a call to *dup2(fildes, fildes2)* returns a value of (*int*)-1 and sets *errno* to [EBADF].

08(A) If a call to *sysconf(_SC_OPEN_MAX)* does not return -1:

When *fildes2* is greater than or equal to {OPEN_MAX}, then a call to *dup2(fildes, fildes2)* returns a value of (*int*)-1 and sets *errno* to [EBADF].

Otherwise:

When *fildes2* takes the value {PCTS_OPEN_MAX}, then a call to *dup2*() does not give an [EBADF] error.

09(B) When *dup2*() is interrupted by a signal, then the call to *dup2(fildes, fildes2)* returns a value of (*int*)-1 and sets *errno* to [EINTR].

See Reason 3 in Section 5. of POSIX.3 {4}.

6.3 File Descriptor Deassignment

6.3.1 *close*()

6.3.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `int close(int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *close*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *close*() is defined as a macro when the header <unistd.h> is included:

When the macro *close*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

03(C) If *close*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

6.3.1.2 Description

04(A) The *close*() function deallocates the file descriptor indicated by its argument (i.e., this file descriptor will be made available for subsequent *open*()s by the process), and a value of zero is returned.

05(A) When a close on the file descriptor is performed, then all outstanding record locks owned by the process on the file associated with the file descriptor are removed.

R01 When *close*() is interrupted by a signal that is to be caught, then the call to *close*() returns a value of (*int*)-1 and sets *errno* to [EINTR]. (See Assertion 13 in 6.3.1.4.)

D01(C) If the state of *fildes* when *close*() is interrupted by a signal that is caught is documented:
The details are contained in 6.3.1.2 of the PCD.1.

06(B) When all file descriptors associated with a pipe have been closed, then data remaining in the pipe is discarded.

See Reason 3 in Section 5. of POSIX.3 [4].

07(A) When all file descriptors associated with a FIFO special file have been closed, then data remaining in the FIFO is discarded.

08(B) When all file descriptors associated with an open file description have been closed, then the open file description is freed.

See Reason 1 in Section 5. of POSIX.3 [4].

09(B) When all file descriptors associated with a file have been closed and the link count of the file is zero, then the space occupied by the file is freed.

See Reason 1 in Section 5. of POSIX.3 [4].

10(B) When all file descriptors associated with a file have been closed and the link count of the file is zero, then the file is no longer accessible.

See Reason 3 in Section 5. of POSIX.3 [4].

11(A) When a file is closed by the last process that had it open, all time-related fields still marked for update are updated.

6.3.1.3 Returns

R02 When a call to *close*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 4 in 6.3.1.2.)

R03 When a call to *close*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertions 12 and 13 in 6.3.1.4.)

6.3.1.4 Errors

12(A) When the *fildes* argument is not a valid file descriptor, then a call to *close*(*fildes*) returns a value of (*int*)-1 and sets *errno* to [EBADF].

13(B) When *close*() is interrupted by a signal, then the call to *close*(*fildes*) returns a value of (*int*)-1 and sets *errno* to [EINTR].

See Reason 3 in Section 5. of POSIX.3 [4].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4 Input and Output

6.4.1 *read*()

6.4.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `ssize_t read(int, void *, size_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *read*() is declared with the result type *ssize_t*. (See GA36 in 2.7.3.)

02(C) If *read*() is defined as a macro when the header <unistd.h> is included:
When the macro *read*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *ssize_t*. (See GA37 in 2.7.3.)

03(C) If *read*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

6.4.1.2 Description

04(A) A call to *read(fildes, buf, nbytes)* reads no more than *nbyte* bytes from the file associated with the open file descriptor *fildes* into the buffer pointed to by *buf* and returns the number of bytes read.

05(A) A call to *read(fildes, buf, 0)* returns zero and does not move the file offset pointer.

06(A) On a regular file, a call to *read(fildes, buf, nbytes)* starts at a position in the file given by the file offset associated with *fildes*.

07(A) Before a call to *read*() returns, the file offset is incremented by the number of bytes actually read.

D01(C) If the value of the file offset, after a *read*() of a file that is not capable of seeking, is documented:
The details are contained in 6.4.1.2 of the PCD.1.

08(A) On a file not capable of seeking, a call to *read*() starts at the current position in the file.

09(A) When a *read*() on a regular file requests more bytes than exist, then the call to *read*() returns the number of bytes from the current position to the end-of-file.

10(A) When a *read*() on a pipe or FIFO requests more bytes than are available and at least one byte is available, then the call to *read*() returns the number of bytes that are available.

Testing Requirements:
Test for both a pipe and a FIFO.

R01 When *read*() is interrupted by a signal before any data is read, then the call to *read*() returns a value of (*int*)-1 and sets *errno* to [EINTR]. (See Assertion 31 in 6.4.1.4.)

D02(C) If the conditions under which *read*(), when interrupted by a signal after having successfully read some data, returns -1 and sets *errno* to [EINTR] or returns the number of bytes read are documented:
The details are contained in 6.4.1.2 of the PCD.1.

11(PCTS_GTI_DEVICE?C:UNTESTED)

If {PCD_READ_INTERRUPTED} is **TRUE**:
When *read*() to a terminal device file is interrupted by a signal after successfully reading some data, then the call to *read*() returns the number of bytes read.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12(PCTS_GTI_DEVICE?C:UNTESTED)

If {PCD_READ_INTERRUPTED} is **FALSE**:
When *read*() to a terminal device file is interrupted by a signal after successfully reading some data, then the call to *read*() returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR].

13(PCTS_GTI_DEVICE?C:UNTESTED)

If {PCD_READ_INTERRUPTED} is not documented:
When *read*() to a terminal device file is interrupted by a signal after successfully reading some data, then the call to *read*() either returns the number of bytes read or returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR].

14(D) If {PCD_READ_INTERRUPTED} is **TRUE** and the implementation supports character special files:
When *read*() to a terminal device file is interrupted by a signal after successfully reading some data, then the call to *read*() returns the number of bytes read.

See Reason 1 in Section 5. of POSIX.3 {4}.

15(D) If {PCD_READ_INTERRUPTED} is **FALSE** and the implementation supports character special files:
When *read*() to a terminal device file is interrupted by a signal after successfully reading some data, then the call to *read*() returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR].

See Reason 1 in Section 5. of POSIX.3 {4}.

16(D) If {PCD_READ_INTERRUPTED} is not documented and implementation supports character special file:
When *read*() to a terminal device file is interrupted by a signal after successfully reading some data, then the call to *read*() either returns the number of bytes read or returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR].

See Reason 1 in Section 5. of POSIX.3 {4}.

17(B) When *read*() is interrupted by a signal after successfully reading some data from a pipe or FIFO, then the call to *read*() returns the number of bytes read.

Testing Requirements:
Test for both a pipe and a FIFO.

See Reason 3 in Section 5. of POSIX.3 {4}.

18(A) When the file offset for a regular file is at or beyond the end-of-file, then a call to *read*() returns zero.

D03(A) When a call to *read*() has returned an end-of-file, the result of a subsequent *read*() request on device special files is contained 6.4.1.2 of the PCD.1.

D04(A) The result of a *read*() with a value of *nbyte* greater than {SSIZE_MAX} is contained in 6.4.1.2 of the PCD.1.

19(A) When no process has an empty pipe or FIFO open for writing, then a call to *read*() on the pipe or FIFO returns zero.

Testing Requirements:
Test for both a pipe and a FIFO.

R02 When some process has an empty pipe or FIFO open for writing and the O_NONBLOCK flag is set, then a call to *read*() on the pipe or FIFO returns a value of (*int*)–1 and sets *errno* to [EAGAIN]. (See Assertion 27 in 6.4.1.4.)

20(A) When some process has an empty pipe or FIFO open for writing and the O_NONBLOCK flag is clear, then a call to *read*() on the pipe or FIFO blocks until some data is written or until the pipe or FIFO is closed by all processes that had it opened for writing.

Testing Requirements:
Test for both a pipe and a FIFO.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

R03 When the O_NONBLOCK flag is set on the file descriptor and the process would be delayed in the read operation, then a call to *read*() for this character special file returns a value of (*int*)-1 and sets *errno* to [EAGAIN]. (See Assertion 22 in 7.1.1.5.)

R04 When the O_NONBLOCK flag is set on the file descriptor and the process would be delayed in the read operation, then a call to *read*() for this special file returns a value of (*int*)–1 and sets *errno* to [EAGAIN]. (See Assertion 28 in 6.4.1.4.)

R05 When the O_NONBLOCK flag is clear on the file descriptor and no data is currently available, then a call to *read*() for this special file blocks until some data becomes available. (See Assertion 19 in 7.1.1.5.)

21(D) If the implementation supports block special files with nonblocking I/O:
When the O_NONBLOCK flag is clear on a block special file descriptor and no data is currently available, then a call to *read*() for this special file blocks until some data becomes available.

See Reason 1 in Section 5. of POSIX.3 {4}.

R06 When the O_NONBLOCK flag is set on the character special file descriptor and data is currently available, then a call to *read*() for this special file returns the available data up to the number of bytes requested. (See Assertions 20 and 21 in 7.1.1.5.)

22(D) If the implementation supports block special files with nonblocking I/O:
When the O_NONBLOCK flag is set on the block special file descriptor and data is currently available, then a call to *read*() for this special file returns the available data up to the number of bytes requested.

See Reason 1 in Section 5. of POSIX.3 {4}.

23(A) When the file pointer precedes the end-of-file and data has not been written to that part of the file, then a call to *read*() from a regular file returns bytes with value zero.

Testing Requirements:
Test using *lseek*() and *fseek*().

24(A) A successful call to *read*() of more than zero bytes marks for update the *st_atime* field of the file.

25(A) A successful call to *read*() with nbytes set to zero does not mark for update the *st_atime* field of the file.

26(A) When *nbyte* is greater than zero, then a successful call to *read*(), which does not read any data from the file, marks for update the *st_atime* field of the file.

6.4.1.3 Returns

R07 When a call to *read*() completes successfully, then the number of bytes read is returned. (See Assertion 4 in 6.4.1.2.)

R08 When a call to *read*() completes unsuccessfully, then a value of (*ssize_t*)-1 is returned and sets *errno* to indicate the error. (See Assertions 27–33 in 6.4.1.4.)

6.4.1.4 Errors

27(A) When the file type is a pipe or FIFO, when the O_NONBLOCK flag is set for the file descriptor, and when the process would be delayed in the read operation, then a call to *read*() returns a value of (*ssize_t*)-1 and sets *errno* to [EAGAIN].

Testing Requirements:
Test for both a pipe and a FIFO.

R09 When the O_NONBLOCK flag is set on a character special file for a file that supports nonblocking I/O, and when the proceeds would be delayed in the read operation, then a call to *read*() for this special file returns a value of (*ssize_t*)-1 and sets *errno* to [EAGAIN]. (See Assertion 21 in 7.1.1.5.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

28(D) If the implementation supports block special files with nonblocking I/O:
When the O_NONBLOCK flag is set on such a block special file descriptor, and when the process would be delayed in the read operation, then a call to *read*() for this special file returns a value of (*ssize_t*)-1 and sets *errno* to [EAGAIN].

See Reason 1 in Section 5. of POSIX.3 {4}.

29(A) When the file descriptor argument *fildes* is not a valid file descriptor, then a call to *read(fildes, buf, nbytes)* returns a value of (*ssize_t*)-1 and sets *errno* to [EBADF].

30(A) When the file descriptor argument *fildes* is not open for reading, then a call to *read(fildes, buf, nbytes)* returns a value of (*ssize_t*)-1 and sets *errno* to [EBADF].

31(A) When *read*() is terminated due to receipt of a signal before any data is read, then the call to *read*() returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR].

D05(C) If the implementation documents whether there are conditions other than those specified by POSIX.1 {3} under which *read*() sets *errno* to [EIO]:
The details are contained in 6.4.1.4 of the PCD.1.

32(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a process in a background process group is attempting to *read*() from its controlling terminal, and when the process is ignoring or blocking the SIGTTIN signal, then the call to *read*() returns a value of (*ssize_t*)-1 and sets *errno* to [EIO].

33(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a process in a background process group is attempting to *read*() from its controlling terminal, and when the process group of the process is orphaned, then the call to *read*() returns a value of (*ssize_t*)-1 and sets *errno* to [EIO].

6.4.2 *write*()

6.4.2.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype
`ssize_t write(int, const void *, size t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *write*() is declared with the result type *ssize_t*. (See GA36 in 2.7.3.)

02(C) If *write*() is defined as a macro when the header <unisrd.h> is included:
When the macro *write*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *ssize_t*. (See GA37 in 2.7.3.)

03(C) If *write*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

6.4.2.2 Description

04(A) A call to *write(fildes, buf, nbyte)* writes *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the open file descriptor argument *fidles* and returns the number of bytes written.

05(A) When the file is a regular file, then a call to *write(times, buf, O)* returns zero, does not mark for update the *st_ctime* and *st_mtime* fields of the file, and no bytes are written.

Testing Requirements:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Test for both O_APPEND flag clear and O_APPEND flag set. The latter case will ensure that an attempt to append zero bytes to the file will not result in a change of the file offset.

D01(C) If the results of a call to *write*() when *nbyte* is zero and the file is not a regular file are documented:
The details are contained in 6.4.2.2 of the PCD.1.

06(A) When the file is a regular file, or another file capable of seeking, then a call to *write(fildes, buf, name)* starts writing at a position in the file given by the file offset associated with the file descriptor argument *fildes.*

07(A) Before a successful return from a call to *write*(), the file offset is incremented by the number of bytes actually written.

08(A) When the file is a regular file, and when the current file offset after a successful return from a call to *write*() is greater than the length of the file before the call, then the length of the file after the call is set to this file offset.

09(A) When the file is not capable of seeking, then a call to *write*() starts at the current position in the file.

D02(C) If the value of the file offset after a call to *write*() for a file type that is not capable of seeking is documented:
The details are contained in 6.4.2.2 of the PCD.1.

10(A) When the O_APPEND flag of the file status flags is set, then a call to *write*() sets the file offset to the end of the file prior to each *write*().

Testing Requirements:
Test for files opened in O_WRONLY and O_RDWR modes.

11(B) When the file is a regular file, and when a *write*() requests that more bytes be written than there is room for, then the call to *write*() writes only as many bytes for which there is room.

See Reason 4 in Section 5. of POSIX.3 {4}.

R01 When *write*() is interrupted by a signal before any data is written, then the call to *write*() returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR]. (See Assertion 39 in 6.4.2.4.)

D03(C) If the conditions under which *write*(), when interrupted by a signal after it has successfully written some data, returns -1 and sets *errno* to [EINTR] or returns the number of bytes read are documented:
The details are contained in 6.4.2.2 of the PCD.1.

12(PCTS_GTI_DEVICE?C:UNTESTED)

If {PCD_WRITE_INTERRUPTED} is **TRUE**:
When *write*() to a terminal device file is interrupted by a signal after successfully writing some data, then the call to *write*() returns the number of bytes written.

13(PCTS_GTI_DEVICE?C:UNTESTED)

If {PCD_WRITE_INTERRUPTED} is **FALSE**:
When *write*() to a terminal device file is interrupted by a signal after successfully writing some data, then the call to *write*() returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR].

14(PCTS_GTI_DEVICE?C:UNTESTED)

If {PCD_WRITE_INTERRUPTED} is not documented:
When *write*() to a terminal device file is interrupted by a signal after successfully writing some data, then the call to *write*() either returns the number of bytes written or returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR].

15(D) If {PCD_WRITE_INTERRUPTED} is **TRUE** and the implementation supports character special files:
When *write*() to a character special file is interrupted by a signal after successfully writing some data, then the call to *write*() returns the number of bytes written.

See Reason 1 in Section 5. of POSIX.3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

16(D) If {PCD WRITE_INTERRUPTED} is **FALSE** and the implementation supports character special files:
When *write*() to a character special file is interrupted by a signal after successfully writing some data, then the call to *write*() returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR].

See Reason 1 in Section 5. of POSIX.3 {4}.

17(D) If {PCD_WRITE_INTERRUPTED} is not documented and the implementation supports character special files:
When *write*() to a character special file is interrupted by a signal after successfully writing some data, then the call to *write*() either returns the number of bytes written or returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR].

See Reason 1 in Section 5. of POSIX.3 {4}.

D04(A) When the value of *nbyte* is greater than {SSIZE_MAX}, the details describing the results of a call to *write*() are contained in 6.4.2.2 of the PCD.1.

18(A) When a *write*() to a regular file has returned successfully, then a successful *read*() from any byte position that was modified by the previous *write*() returns the data written to that position by that previous *write*() until such byte positions are again modified.

19(A) When a regular file already contains data at the position referenced by a successful call to *write*(), then the data at the position referenced are overwritten.

20(B) When the file is a pipe or a FIFO, when *write*()has transferred some data, and when *nbyte* is less than or equal to {PIPE_BUF}, then the call to *write(fildes, buf, nbyte)* does not return with *errno* set to [EINTR].

Testing Requirements:
Test for both a pipe and a FIFO.

See Reason 2 in Section 5. of POSIX.3 {4}.

21(A) When the file is a pipe or a FIFO, then a call to *write*() appends to the end of the pipe or FIFO.

Testing Requirements:
Test for both a pipe and a FIFO.

22(B) When the file is a pipe or a FIFO and *nbyte* is less than or equal to {PIPE_BUF}, then the call to *write*() does not interleave with data from other processes doing writes on the same pipe or FIFO.

Testing Requirements:
Test for both a pipe and a FIFO.

See Reason 3 in Section 5. of POSIX.3 {4}.

R02 When the file is a pipe or a FIFO, when the O_NONBLOCK flag is set, and when no data can be accepted at the time of the *write*(), then the call to *write*() does not block the process. (See Assertion 28 in 6.4.2.2.)

23(D) If the implementation supports character special files with nonblocking I/O:
When the file is a character special file, when the O_NONBLOCK flag is set, and when no data can be accepted at the time of the *write*(), then the call to *write*() does not block the process.
NOTE — The case of a terminal device file is covered by Assertion 43 in 7.1.1.8.

See Reason 1 in Section 5. of POSIX.3 {4}.

24(D) If the implementation supports block special files with nonblocking I/O:
When the file is a block special file, when the O_NONBLOCK flag is set, and when no data can be accepted at the time of the *write*(), then the call to *write*() does not block the process.

See Reason 1 in Section 5. of POSIX.3 {4}.

25(A) When the file is a pipe or a FIFO, when the O_NONBLOCK flag is set, when *nbyte* is less than or equal to {PIPE_BUF}, and when space exists in the pipe or FIFO for *nbyte* bytes of data, then a call to *write(fildes, buf, nbyte)* succeeds completely and returns *nbyte*.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Testing Requirements:

Test for both a pipe and a FIFO. If {PIPE_BUF} is greater than {PCTS_PIPE_BUF}, test with values of *nbyte* up to and including {PCTS_PIPE_BUF}.

26(A) When the file is a pipe or a FIFO, when the O_NONBLOCK flag is clear, and when *nbyte* is less than or equal to {PIPE_BUF}, then a call to *write(fildes, buf, nbyte)* blocks until space is available to complete the *write*() or the *write*() is interrupted by a signal.

Testing Requirements:

If {PIPE_BUF} is greater than {PCTS_PIPE_BUF}, test with values of *nbyte* up to and including {PCTS_PIPE_BUF}.

27({PIPE_BUF} ≤ {PCTS_PIPE_BUF}?A:UNTESTED)

When the file is a pipe or a FIFO, when the O_NONBLOCK flag is set, when *nbyte* is greater than {PIPE_BUF} bytes, and when at least one byte can be written, then a call to *write*() transfers what it can and returns the number of bytes written.

Testing Requirements:

Test for both a pipe and a FIFO.

28({PIPE_BUF} ≤ {PCTS_PIPE_BUF}?A:UNTESTED)

When the file is a pipe or a FIFO, when the O_NONBLOCK flag is set, when *nbyte* is greater than {PIPE_BUF} bytes, and when no data can be written, then a call to *write*() returns a value of (*ssize_t*)-1, sets *errno* to [EAGAIN], and transfers no data.

Testing Requirements:

Test for both a pipe and a FIFO.

29({PIPE_BUF} ≤ {PCTS_PIPE_BUF}?A:UNTESTED)

When the file is a pipe or a FIFO, when the O_NONBLOCK flag is set, when *nbyte* greater than {PIPE_BUF} bytes, and when all data previously written to the pipe or FIFO has been read, then a call to *write(fildes, buf, nbyte)* transfers at least {PIPE_BUF} bytes.

Testing Requirements:

Test for both a pipe and a FIFO.

30(D) If the implementation supports character special files with nonblocking I/O:

When the file is a character special file, when the O_NONBLOCK flag is clear, and when the file cannot accept the data immediately, then a call to *write*() blocks until the data can be accepted.

NOTE — The case of a terminal device file is covered by Assertion 43 in 7.1.1.8.

See Reason 1 in Section 5. of POSIX.3 [4].

31(D) If the implementation supports block special files with nonblocking I/O:

When the file is a block special file, when the O_NONBLOCK flag is clear, and when the file cannot accept the data immediately, then a call to *write*() blocks until the data can be accepted.

See Reason 1 in Section 5. of POSIX.3 [4].

32(D) If the implementation supports character special files with nonblocking I/O:

When the file is a character special file, when the O_NONBLOCK flag is set, and when some data can be written without blocking the process, then a call to *write*() either writes what it can and returns the number of bytes written, or returns a value of (*ssize_t*)-1, sets *errno* to [EAGAIN], and transfers no data.

NOTE — The case of a terminal device file is covered by Assertion 44 in 7.1.1.8.

See Reason 1 in Section 5. of POSIX.3 [4].

33(D) If the implementation supports block special files with nonblocking I/O:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the file is a block special file, when the O_NONBLOCK flag is set, and when some data can be written without blocking the process, then a call to *write*() either writes what it can and returns the number of bytes written, or returns a value of (*ssizet*)-1, sets *errno* to [EAGAIN], and transfers no data.

See Reason 1 in Section 5. of POSIX.3 {4}.

34(A) A successful call to *write*() of more than zero bytes marks for update the *st_ctime* and *st_mtime* fields of the file.

6.4.2.3 Returns

R03 When a call to *write*() completes successfully, then the number of bytes written is returned. (See Assertion 4 in 6.4.2.2.)

R04 When a call to *write*() completes unsuccessfully, then a value of (*ssize_t*)-1 is returned and sets *errno* to indicate the error. (See Assertions 35–42 in 6.4.2.4.)

6.4.2.4 Errors

35(A) When the file is a pipe or a FIFO, when the O_NONBLOCK flag is set, when *nbyte* is less than or equal to {PIPE_BUF}, and when insufficient capacity exists to accept the data, then a call to *write*() returns a value of (*ssize_t*)-1 and sets *errno* to [EAGAIN].

Testing Requirements:

Test for both a pipe and a FIFO. If {PIPE_BUF} is greater than {PCTS_PIPE_BUF}, test with values of *nbyte* up to and including {PCTS_PIPE_BUF}.

36(A) When the file descriptor argument *fildes* is not a valid file descriptor, then a call to *write*(*fildes, buf, nbyte*) returns a value of (*ssize_t*)-1 and sets *errno* to [EBADF].

37(A) When the file descriptor argument *fildes* is not open for writing, then a call to *write*(*fildes, buf, nbyte*) returns a value of (*ssize_t*)-1 and sets *errno* to [EBADF].

38(D) If the implementation supports a maximum file size:

When an attempt is made to write a file that would exceed an implementation-defined maximum size, then a call to *write*() returns a value of (*ssize_t*)-1 and sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX.3 {4}.

39(A) When *write*() is terminated due to receipt of a signal and no data was transferred, then the call to *write*() returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR].

R05 When *write*() is interrupted by a signal after successfully writing some data, then the call to *write*() either returns the number of bytes written or returns a value of (*ssize_t*)-1 and sets *errno* to [EINTR]. (See Assertions 12–17 in 6.4.2.2.)

D05(C) If the implementation documents whether there are conditions other than those specified by POSIX.1 {3} under which *write*() sets *errno* to [EIO]:

The details are contained in 6.4.2.4 of the PCD.1.

40(PCTS_GTI_DEVICE:C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process is a member of an orphaned background process group attempting to *write*() to its controlling terminal, when TOS-TOP is set, and when the process is neither ignoring nor blocking the SIGTTOU signal, then a call to *write*() returns a value of (*ssize_t*)-1 and sets *errno* to [EIO].

41(B) When a device has no more space for data, then a call to *write*() on this device returns a value of (*ssize_t*)-1 and sets *errno* to [ENOSPC].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

See Reason 4 in Section 5. of POSIX.3 {4}.

42(A) When the file is a pipe or a FIFO and the file is not open for reading by any process, then a call to *write*() returns a value of (*ssize_t*)-1, sets *errno* to [EPIPE], and sends a SIGPIPE signal to the calling process.

Testing Requirements:

Test for both a pipe and a FIFO.

6.5 Control Operations on Files

6.5.1 Data Definitions for File Control Operations

01(A) When the header <fcntl.h> is included, then the symbolic constants O_CREAT, O_EXCL, O_TRUNC, O_NOCTTY, O_APPEND, O_NONBLOCK, O_RDONLY, O_RDWR, O_WRONLY, and O_ACCMODE are defined.

02(A) When the header <fcntl.h> is included, then the symbolic constants F_DUPFD, F_GETFD, F_GETLK, F_SETFD, F_GETFL, F_SETFL, F_SETLK, and F_SETLKW are defined, and their values are unique, numbers.

03(A) When the header <fcntl.h> is included, then the symbolic constant FD_CLOEXEC is defined.

04(A) When the header <fcntl.h> is included, then the symbolic constants F_RDLCK, F_UNLCK, and F_WRLCK are defined, and their values are unique numbers.

05(A) When the header <fcntl.h> is included, then the set of symbolic constants O_RDONLY, O_WRONLY and O_RDWR are defined, and their values are unique.

06(A) When the header <fcntl.h> is included, then the set of symbolic constants O_ACCMODE, O_APPEND, O_CREAT, O_EXCL, O_NONBLOCK, O_NOCTTY, and O_TRUNC are defined, and their values are bitwise discrete.

07(A) When the header <fcntl.h> is included, then O_ACCMODE is defined as a mask for O_RDONLY, O_RDWR, and O_WRONLY.

6.5.2 *fcntl*()

6.5.2.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <fcntl.h> is included, then the function prototype `int fcntl(int, int, ...)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <fcntl.h> is included, then the function *fcntl*() is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *fcntl*() is defined as a macro when the header <fcntl.h> is included:

When the macro *fcntl*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *fcntl*() is defined as a macro in the header <fcntl.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

6.5.2.2 Description

04(A) A call to *fcntl*(*fildes, F_DUPFD, arg*) provides the following attributes for the new file descriptor created:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— It is the lowest numbered available file descriptor greater than or equal to argument *arg*.
— It refers to the same open file description (file pointer, access mode, file status flags) as the original file descriptor.
— It shares the same locks as the original file descriptor.
— It has the FD_CLOEXEC flag clear.

05(A) A call to *fcntl(fildes, F_GETFD)* returns the status of the file descriptor flags.

06(A) When the flag in *arg* corresponding to FD_CLOEXEC is nonzero, then a call to *fcntl(fildes, F SETFD, arg)* sets the close-on-exec flag, and the file associated with *fildes* is closed after a successful exec.

07(A) When the flag in *arg* corresponding to FD_CLOEXEC is zero, then a call to *fcntl(fildes, F SETFD, arg)* clears the close-on-exec flag, and the file associated with *fildes* stays open across exec type functions.

08(A) A call to *fcntl(fildes, F_GETFL)* returns the file status flags O_APPEND and O_NONBLOCK, and the file access modes O_RDONLY, O_RDWR, and O_WRONLY.

09(A) A call to *fcntl(fildes, F_SETFL, O)* sets the status flags to 0.

10(A) A call to *fcntl(fildes, F. SETFL, O_NONBLOCK)* sets the status flags to O_NONBLOCK.

11(A) A call to *fcntl(fildes, F_SETFL, O_APPEND)* sets the status flags to O_APPEND.

12(A) A call to *fcntl(fildes, F SETFL, O_APPEND | O_NONBLOCK)* sets the status flags for O_APPEND and O_NONBLOCK.

13(A) When the third argument *arg* only contains the file status flags O_APPEND, O_NONBLOCK; the file access modes O_RDONLY, O_RDWR, O_WRONLY; and the *oflag* values O_CREAT, O_EXCL, O_NOCTTY, and O_TRUNC; then a call to *fcntl(fildes, F_SETFL, arg)* ignores any bits corresponding to the file access modes and the *oflag* values that are set in *arg*.

D01(C) If the implementation supports additional file status flags for the F_SETFL command for *fcntl*(), and this is documented:

The details on these additional file status flags are contained in 6.5.2.2 of the PCD.1.

D02(C) If advisory record locking is supported or unsupported for files other than regular files, and this is documented:

The details are contained in 6.5.2.2 of the PCD.1.

14(A) When a lock is present that blocks the request and *arg* is a pointer to the type *struct flock*, then a call to *fcntl(fildes, F_GETLK, arg)* returns the first such lock that blocks the lock description pointed to by arg. All elements of the *struct flock* addressed by *arg* are set, and the lock type is placed in the *struct flock* addressed by *arg* with the value of *l_whence* set to SEEK_SET.

Testing Requirements:

Test for all three values of *l_whence*.

15(A) When no lock exists in the region specified in the *struct flock* addressed by *arg* that would prevent the lock from being created, then a call to *fcntl(fildes, F_GETLK, arg)* sets the lock type in the structure pointed to by *arg* to F_UNLCK and does not change the other *struct flock* elements.

16(A) When the lock type in the structure pointed to by *arg* is set to F_RDLCK, then a call to *fcntl(fildes, F_SETLK, arg)* establishes a shared lock for the region specified in the *struct flock* addressed by *arg*.

17(A) When the lock type in the structure pointed to by *arg* is set to F_WRLCK, then a call to *fcntl(fildes, F_SETLK, arg)* establishes an exclusive lock for the region specified in the *struct flock* addressed by *arg*.

18(A) When the lock type in the structure pointed to by *arg* is set to F_UNLCK, then a call to *fcntl(fildes, F SETLK, arg)* removes any locks for the region specified in the *struct flock* addressed by *arg*.

19(A) When a shared or exclusive lock cannot be set, then a call to *fcntl(fildes, F_SETLK, arg)* returns without waiting.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

20(A) When lock type in the structure pointed to by *arg* is set to F_RDLCK, then a call to *fcntl(fildes, F_SETLKW, arg)* establishes a shared lock for the region specified in the *struct flock* addressed by *arg*.

21(A) When the lock type in the structure pointed to by *arg* is set to F_WRLCK, then a call to *fcntl(fildes, F_SETLKW, arg)* establishes an exclusive lock for the region specified in the *struct flock* addressed by *arg*.

22(A) When the lock type in the structure pointed to by *arg* is set to F_UNLCK, then a call to *fcntl(fildes, F_SETLKW, arg)* removes any locks for the region specified in the *struct flock* addressed by *arg*.

23(A) When the shared or exclusive lock is blocked by other locks, then a call to *fcntl(fildes, F_SETLKW, arg)* waits until the request can be satisfied.

R01 When *fcntl()* is interrupted while waiting to set a lock on a region, then a call to *fcntl(fildes, F_SETLKW, arg)* returns a value of (*int*)-1 and sets *errno* to [EINTR]. (See Assertion 40 in 6.5.2.4.)

24(A) When the header `<fcntl.h>` is included, then the *struct flock* is defined and has the element names and types specified in Table 6.1.

Table 6.1—*flock* Structure

Element	Type
l_type	short
l_whence	short
l-start	off_t
l_len	off_t
l_pid	pid_t

25(A) When a shared lock has been set on a segment of a file, then other processes are able to set shared locks on that segment or a portion of it.

26(A) A shared lock prevents any other process from setting an exclusive lock on any portion of the part of the file to which the shared lock is applied.

R02 A request for a shared lock will fail if the file descriptor was not opened with read access. (See Assertion 38 in 6.5.2.4.)

27(A) An exclusive lock prevents any other process from setting a shared lock or an exclusive lock on any portion of the protected area.

R03 A request for an exclusive lock fails if the file descriptor was not opened with write access. (See Assertion 39 in 6.5.2.4.)

D03(C) If the action taken on calls to *fcntl()* for file locking when *l_len* is negative is documented:
The details are contained in 6.5.2.2 of the PCD.1.

R04 The *l_pid* field is only used with F_GETLK to return the process ID of the process holding a blocking lock. (See Assertion 14 in 6.5.2.2.)

28(A) Locks can start and extend beyond the current end-of-file.

29(A) When *l_len* is set to zero, then all bytes from the position specified by *l_whence* and *l_start* up to the end-of-file irrespective of the current file length are locked or unlocked as specified by the structure pointed to by *arg*.

30(A) Before a successful return from a F_SETLK or F_SETLKW request, the previous lock type for each byte in the specified region is replaced by the new lock type. The calling process has only one type of lock set for each byte in the file.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

31(A) All locks associated with a file for a given process areremoved when a file descriptor for that file is closed.

32(A) When a process terminates, then all locks associated with a file for the process are removed.

33(A) Locks are not inherited by a child process created using the *fork*() function.

6.5.2.3 Returns

R05 When a call to *fcntl(fildes, F_DUPFD, arg)* completes successfully, then the lowest file descriptor greater than or equal to arg is returned. (See Assertion 4 in 6.5.2.2.)

R06 When a call to *fcntl(fildes, F_GETFD)* completes successfully, then the value of the close_on_exec flag is returned. (See Assertion 5 in 6.5.2.2.)

R07 When *cmd* are the command values F_SETFD, F_GETFL, F_SETFL, F_GETLK, F_SETLK, and F_SETLKW, and when the call to *fcntl(fildes, cmd)* completes successfully, then a value other than -1 is returned. (See Assertions 6–23 in 6.5.2.2.)

R08 When a call to *fcntl*() completes unsuccessfully, then a value of (*int*)–1 is returned and sets *errno* to indicate the error. (See Assertions 34–52 in 6.5.2.4.)

6.5.2.4 Errors

34(A) When the locking structure pointed to by *arg* contains F_RDLCK, and when some portion of the segment of a file to be locked is already exclusive-locked by another process, then a call to *fcntl(fildes, F_SETLK, arg)* returns a value of (*int*)-1 and sets *errno* to [EACCES] or [EAGAIN].

35(A) When the locking structure pointed to by *arg* contains F_WRLCK, and when some portion of the segment of a file to be locked is already exclusive-locked by another process, then a call to *fcntl(fildes, F_SETLK, arg)* returns a value of (*int*)-1 and sets *errno* to [EACCES] or [EAGAIN].

36(A) When the locking structure pointed to by arg contains F_WRLCK, and when some portion of the segment of the file to be locked is already shared-locked by another process, then a call to *fcntl(fildes, F_SETLK, arg)* returns a value of (*int*)-1 and sets *errno* to [EACCES] or [EAGAIN].

37(A) When the file descriptor argument is not a valid file descriptor, then a call to *fcntl(fildes, cmd, arg)* returns a value of (*int*)-1 and sets *errno* to [EBADF].

38(A) When the type of lock is a F_RDLCK lock but *fildes* is not a valid file descriptor open for reading, then a call to *fcntl(fildes, F_SETLK, arg)* and *fcntl(fildes, F_SETLKW, arg)* returns a value of (*int*)-1 and sets *errno* to [EBADF].

39(A) When the type of lock is a F_WRLCK lock but *fildes* is not a valid file descriptor open for writing, then a call to *fcntl(fildes, F_SETLK, arg)* and *fcntl(fildes, F_SETLKW, arg)* returns a value of (*int*)-1 and sets *errno* to [EBADF].

40(A) When *fcntl*() is interrupted by a signal, then a call to *fcntl(fildes, F_SETLKW, arg)* returns a value of (*int*)-1 and sets *errno* to [EINTR].

41(A) When *arg* is negative, then a call to *fcntl(fildes, F_DUPFD, arg)* returns a value of (*int*)–1 and sets *errno* to [EINVAL].

42(A) If a call to *sysconf(_SC_OPEN_MAX)* does not return -1:

When *arg* is greater than or equal to {OPEN_MAX}, then a call to *fcntl(fildes, F_DUPFD, arg)* returns a value of (*int*)-1 and sets *errno* to [EINVAL].

Otherwise:

When *arg* takes the value {PCTS_OPEN_MAX}, then a call to *fcntl(fildes, F_DUPFD, arg)* does not give an [EINVAL] error.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

43(A) When the data to which *arg* points references a position prior to the beginning of the file, then a call to *fcntl(fildes, F_GETLK, arg)*, *fcntl(fildes, F_SETLK, arg)*, or *fcntl(fildes, F_SETLKW, arg)* returns a value of (*int*)–1 and sets *errno* to [EINVAL].

44(A) When the *l_type* field of the structure pointed to by *arg* is invalid, then a call to *fcntl(fildes, F_GETLK, arg)*, *fcntl(fildes, F_SETLK, arg)*, or *fcntl(fildes, F_SETLKW, arg)* returns a value of (*int*)-1 and sets *errno* to [EINVAL].

45(A) When the *l_whence* field of the structure pointed to by *arg* is invalid, then a call of *fcntl(fildes, F_GETLK, arg)*, *fcntl(fildes, F_SETLK, arg)*, or *fcntl(fildes, F_SETLKW, arg)* returns a value of (*int*)-1 and sets *errno* to [EINVAL].

46(C) If {PCD_NO_LOCK_FILE TYPE} is **TRUE**:

When the *fildes* refers to a file that does not support locking, then a call of *fcntl(fildes, F_GETLK, arg)*, *fcntl(fildes, F_SETLK, arg)*, or *fcntl(fildes, F_SETLKW, arg)* returns a value of (*int*)-1 and sets *errno* to {EINVAL].

47(C) If {PCD_NO_LOCK_FILE_TYPE} is not documented:

When the lock type in the structure pointed to by arg is set to F_RDLCK, then a call to *fcntl(fildes, F_SETLK, arg)* either establishes a shared lock for the region specified in the *struct flock* addressed by arg or returns a value of (*int*)-1 and sets *errno* to [EINVAL].

Testing Requirements:

Test for file types of FIFO and character special files.

48(A) If {OPEN_MAX} ≤ {PCTS_OPEN_MAX}:

When {OPEN_MAX} files have been opened, then a subsequent call to *fcntl(fildes, F_DUPFD, arg)* returns a value of (*int*)-1 and sets *errno* to [EMFILE].

Otherwise:

{PCTS_OPEN_MAX} files can be opened.

49(A) When no file descriptor greater than or equal to *arg* is available, then a call of *fcntl(fildes, F_DUPFD, arg)* returns a value of (*int*)-1 and sets *errno* to [EMFILE].

50(B) When satisfying the lock or unlock request that would result in the number of locked regions exceeding a system-imposed limit, then a call to *fcntl(fildes, F_SETLK, arg)* or *fcntl(fildes, F_SETLKW, arg)* returns a value of (*int*)-1 and sets *errno* to [ENOLCK].

See Reason 3 in Section 5. of POSIX.3 {4}.

D04(C) If the implementation supports the detection of [EDEADLK] for *fcntl*():

The details under which [EDEADLK] occurs for *fcntl*() are contained in 6.5.2.4 of the PCD.1. (See DGA02 in 2.4.)

51(C) If the implementation supports the detection of [EDEAELK] for *fcntl*():

When a deadlock condition occurs, then a call to *fcntl(fildes, F_SETLKW, arg)* returns a value of (*int*)-1 and sets *errno* to [EDEADLK].

52(D) If the implementation does not support the detection of [EDEADLK] for *fcntl*():

A call to *fcntl(fildes, F_SETLKW, arg)* is successful (unless adifferent error condition is detected). (See GA26 in 2.4.)

See Reason 2 in Section 5. of POSIX.3 {4}.

6.5.3 *lseek* ()

6.5.3.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the header <unistd.h> is included, then the function prototype `off_t lseek(int, off_t, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *lseek*() is declared with the result type *off_t*. (See GA36 in 2.7.3.)

02(C) If *lseek*() is defined as a macro when the header <unistd.h> is included:

When the macro *lseek*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *off_t*. (See GA37 in 2.7.3.)

03(C) If *lseek*() is defined as a macro in the header <unistd.h>:

It evaluates its arguments only once, fully protected by parentheseswhen necessary, and protects its result value with extraparentheses when necessary. (See GA01 in 1.3.4.)

6.5.3.2 Description

04(A) When the file is capable of seeking, then a call to *lseek*(*fildes*, *offset*,*SEEK_SET*) sets the file *offset* offset bytes from the beginning of the file.The resulting offset location as measured in bytes from the beginning ofthe file is returned.

05(A) When the file is capable of seeking, then a call to *lseek*(*fildes*, *offset*,*SEEK_CUR*) sets the file offset offset bytes from the current position.The resulting offset location as measured in bytes from the beginning ofthe file is returned.

06(A) When the file is capable of seeking, then a call to *lseek*(*fildes*, *offset*, *SEEK_END*) sets the file offset to the size of the file plus *offset* bytes. The resulting offset location as measured in bytes from the beginning of the file is returned.

07(A) When the header <unistd.h> is included, then the symbolic constants SEEK_SET, SEEK_CUR, and SEEK_END are defined.

D01(A) The details on the behavior of the *lseek*() function on devices incapable of seeking are contained in 6.5.3.2 of the PCD.1.

R01 When the file is capable of seeking, and when the file offset has been set beyond the end of existing data in the file by a call to *lseek*() followed by a call to *write*(), then data read from gaps in the file—areas in the file where no data has been written—appear as though bytes with the value zero had been written. (See Assertion 18 in 6.4.1.2.)

08(A) When the file is capable of seeking, then a call to *lseek*() does not, by itself, extend the size of the file.

6.5.3.3 Returns

R02 When a call to *lseek*() completes successfully, then the resulting offset location as measured in bytes from the beginning of the file is returned. (See Assertions 4-6 in 6.5.3.2.)

R03 When a call to *lseek*() completes unsuccessfully, then a value of (*off t*)-1 is returned, sets *errno* to indicate the error, and the file pointer remains unchanged. (See Assertions 9–13 in 6.5.3.4.)

6.5.3.4 Errors

09(A) When the file descriptor argument is not a valid file descriptor, then a call to *lseek*(*fildes*, *offset*, *whence*) returns a value of (*off_t*)-1 and sets *errno* to [EBADF].

10(A) When the file is capable of seeking and the *whence* argument is not valid, then a call to *lseek*(*fildes*, *offset*, *whence*) returns a value of (*off_t*)-1, sets *errno* to [EINVAL], and does not alter the value of the file pointer.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11(A) When the file is capable of seeking and the resulting file offset would be invalid, then a call to *lseek(fildes, offset, whence)* returns a value of (*off_t*)-1, sets *errno* to [EINVAL], and does not alter the value of the file pointer.

12(A) When the *fildes* argument is associated with a pipe, then a call to *lseek(fildes, offset, whence)* returns a value of (*off_t*)-1, sets *errno* to [ESPIPE], and does not alter the value of the file pointer.

13(A) When the *fildes* argument is associated with a FIFO then a call to *lseek(fildes, offset, whence)* returns a value of (*off_t*)-1, sets *errno* to [ESPIPE], and does not alter the value of the file pointer.

7. Device- and Class-Specific Functions

Testing Requirements:

The assertions in this section are required to be tested on any device that supports the general terminal interface. This includes devices that control asynchronous communications ports and may include devices that control network connections or synchronous ports. An implementation need not provide any device that supports the general terminal interface, in which case the tests for these assertions will not be executed.
Many of the assertions in this chapter reference particular flags associated with part of the *termios* structure. These references do not explicitly state that these flags need to be set in the *termios* control structure that is in effect for the terminal device file that the test is using to verify the assertion. Unless otherwise stated in the assertion, this shall be the meaning associated with any statement in the assertion regarding the setting of flags contained in the *termios* structure.

7.1 General Terminal Interface

D01(A) The device types supported by the general terminal interface and whether network connections and asynchronous ports or both are supported by the general terminal interface are contained in 7.1 of the PCD.1.

7.1.1 Interface Characteristics

7.1.1.1 Opening a Terminal Device File

There are no assertions specific to this subclause.

7.1.1.2 Process Groups

R01 When all the processes in the foreground process group have terminated, the system does not change an existing background process group to be the foreground process group, but allows the terminal to have no foreground process group. (See Assertion 5 in 7.2.3.3.)

D02(C) If it is documented whether the terminal has a foreground process group when there is no longer any process whose process group ID matches the process group ID of the foreground process group, but there is a process whose process ID matches:
The details are contained in 7.1.1.2 of the PCD.1.

7.1.1.3 The Controlling Terminal

01(PCTS_GTI_DEVICE?A:UNTESTED)

Each process of a session that has a controlling terminal has the same controlling teminal.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

02(PCTS_GTI_DEVICE?B:UNTESTED)

A controlling terminal associated with the current session cannot become the controlling terminal for another session.

See Reason 1 in Section 5. of POSIX.3 [4].

03(PCTS_GTI_DEVICE?A:UNTESTED)

When a process that is not a session leader *open*()s a terminal file, then the terminal does not become the controlling terminal of the calling process.

04(PCTS_GTI_DEVICE?A:UNTESTED)

When a process uses the O_NOCTTY option on a call to *open*(), then the terminal does not become the controlling terminal of the calling process.

D03(A) When a session leader without a controlling terminal opens a terminal device file that is not already associated with a session—without using the O_NOCTTY option—the details on whether the terminal becomes the controlling terminal of the session leader are contained in 7.1.1.3 of the PCD.1.

05(PCTS_GTI_DEVICE?A:UNTESTED)

When a controlling terminal becomes associated with a session, then its foreground process group is set to the process group of the session leader.

R02 The controlling terminal is inherited by child processes during a *fork*() function. (See Assertion 20 in 3.1.1.2.)

06(PCTS_GTI_DEVICE?A:UNTESTED)

When a process creates a new session with the *setsid*() function, then it relinquishes its controlling terminal.

D04(C) If it is documented whether, upon the close of the last file descriptor in the system associated with the controlling terminal, all processes that have that terminal as their controlling terminal cease to have a controlling terminal:

The details are contained in 7.1.1.3 of the PCD.1.

07(PCTS_GTI_DEVICE?A:UNTESTED)

When a process closes all file descriptors associated with its controlling terminal, and when at least one other process has an open file descriptor that references the same controlling terminal, then the first process does not relinquish its controlling terminal.

D05(C) If it is documented whether a session leader can reacquire a controlling terminal after it has relinquished its controlling terminal because all file descriptors associated with that terminal have been closed:

The details are contained in 7.1.1.3 of the PCD.1.

08(PCTS_GTI_DEVICE?A:UNTESTED)

When a controlling process terminates, then the controlling terminal is disassociated from the current session, allowing it to be acquired by a new session leader.

09(PCTS_GTI_DEVICE?C:UNTESTED)

If, after a controlling process terminates, access to the terminal by processes in the session of the terminated process is denied:

Attempts to access the terminal by surviving processes in that session are treated as if modem disconnect had occurred.

Testing Requirements:

Test when *read*() returns 0 and *write*() returns -1 and sets *errno* to [EIO].

D06(C) If it is documented whether, after a controlling process terminates, access to the terminal by processes in the session of the terminated process is denied:

The details are contained in 7.1.1.3 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D07(A) The details how the controlling terminal for a session is allocated by the session leader are contained in 7.1.1.3 of the PCD.1.

7.1.1.4 Terminal Access Control

R03 A process in a foreground process group is allowed to read from its controlling terminal. (See Assertions 19-28 in 7.1.1.5 through 7.1.1.7.)

10(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a member of a background process group that is not orphaned and is not blocking or ignoring the SIGTTIN signal reads from its controlling terminal, then a SIGTTIN signal is sent to each process in its process group.

11(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a member of a background process group that is ignoring the SIGTTOU signal attempts to read from its controlling terminal, then a call to *read*() returns a value of (*ssize_t*)-1, sets *errno* to [EIO], and does not send the SIGTTIN signal to any process in its process group.

12(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a member of a background process group that is blocking the SIGTTIN signal attempts to read from its controlling terminal, then the call to *read*() returns a value of (*ssize_t*)-1, sets *errno* to [EIO], and does not send the SIGTTIN signal to any process in its process group.

13(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} :is supported:
When a member of an orphaned background process group attempts to read from its controlling terminal, then the call to *read*() returns a value of (*ssize_t*)-1, sets *errno* to [EIO], and does not send the SIGTTIN signal to any process in its process group.

14(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB CONTROL} is supported:
When TOSTOP is set, and when a member of a background process group that is not orphaned and is not blocking or ignoring the SIGTTOU signal writes to its controlling terminal, then a SIGTTOU signal is sent to each process in its process group.

15(PCTS_GTI_DEVICE?C :UNTESTED)

If the behavior associated with {_POSIX_JOB CONTROL} is supported:
When TOSTOP is clear, and when a member of a background process group, whether orphaned or not, attempts to write to its controlling terminal, then the call to *write*() succeeds, and the SIGTTOU signal is not sent to any process in its process group.

16(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When TOSTOP is set, and when a member of a background process group that is ignoring the SIGTTOU signal attempts to write to its controlling terminal, then the call to *write*() succeeds.

17(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When TOSTOP is set, and when a member of a background process group that is blocking the SIGTTOU signal attempts to write to its controlling terminal, then the call to *write*() succeeds.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18(PCTS_GTI_DEVICE?C :UNTESTE D)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When TOSTOP is set, and when a process that is a member of an orphaned background process group and is neither ignoring nor blocking SIGTTOU attempts to write to its controlling terminal, then the call to *write*() returns a value of (*ssize_t*)-1, sets *errno* to [EIO], and does not send the SIGTTOU signal to any process in its process group.

7.1.1.5 Input Processing and Reading Data

D08(C) If the system imposes a limit, {MAX_INPUT}, on the number of bytes that may be stored in a terminal input queue:

The behavior of the system when that limit is exceeded is documented in 7.1.1.5 of the PCD.1.

19(PCTS_GTI_DEVICE?A:UNTESTED)

When O_NONBLOCK is clear, then a call to *read*() is blocked until data is available or a signal has been received.

20(PCTS_GTI_DEVICE?A:UNTESTED)

When O_NONBLOCK is set, and when there is enough data available to satisfy the entire request, then a call to *read*() completes successfully, without blocking, reading all the requested data and returning the number of bytes read.

Testing Requirements:

Test for both canonical and noncanonical mode input processing.

21(PCTS_GTI_DEVICE?A:UNTESTED)

When O_NONBLOCK is set, and when there is not enough data available to satisfy the entire request, then a call to *read*() completes successfully, without blocking, reading as much data as possible, and returns the number of bytes it was able to read.

Testing Requirements:

Test for both canonical and noncanonical mode input processing.

22(PCTS_GTI_DEVICE?A:UNTESTED)

When O_NONBLOCK is set and there is no data available, then a call to *read*() completes without blocking, returns a value of (*ssize_t*)-1, and sets *errno* to [EAGAIN].

Testing Requirements:

Test for both canonical and noncanonical mode input processing.

7.1.1.6 Canonical Mode Input Processing

23(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is set and O_NONBLOCK is not set, then a call to *read*() from a terminal does not return until an entire line has been typed or a signal has been received. A line is delimited by NL, EOF, or EOL.

Testing Requirements:

Test for NL, EOF, and EOL.

24(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is set, then a call to *read*() from a terminal returns at most one line, even if more bytes are requested.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

25(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is set, then a call to *read*() requesting fewer bytes than available in the input line receives the number of bytes requested. The first byte received from the next *read*() is the first unread byte in the line, and no input data is lost.

26(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is set, then the canonical input queue of a terminal is able to contain a minimum number of bytes.

Testing Requirements:

When {MAX_CANON} is greater than {PCTS_MAX_CANON}, then the minimum number of bytes the canonical queue can contain is {PCTS_MAX_CANON}; otherwise, it can contain {MAX_CANON} bytes.

D09(C) If the system imposes a limit, {MAX_CANON}, on the number of bytes in a line for a particular terminal device:

The behavior of the system when the terminal is processing input in canonical mode and the limit is exceeded is documented in 7.1.1.6 of the PCD.1.

R04 Erase and kill processing occurs when either of the two special characters, ERASE and KILL, is received. (See Assertions 52 and 53 in 7.1.1.9.)

7.1.1.7 Noncanonical Mode Input Processing

27(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, then all bytes input from a terminal are entered into the input queue and are not assembled into lines.

28(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, then erase and kill processing do not occur. ERASE and KILL are entered into the input queue.

D10(C) If the response to *read*() when MIN is greater than {MAX_INPUT} is documented:

The details are contained in 7.1.1.7 of the PCD.1.

7.1.1.7.1 CASE A: MIN > 0, TIME > 0

29(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is greater than 0, and TIME is greater than 0, then TIME acts as an interbyte timer of 0.1 s granularity, activated after the first byte is received.

30(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is greater than 0, and TIME is greater than 0, then the interbyte timer is reset after each byte is received.

31(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is greater than 0, TIME is greater than 0, and when MIN bytes are received before the interbyte timer expires, then the *read*() is satisfied.

32(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is greater than 0, TIME is greater than 0, and when the interbyte timer expires before MIN bytes are received, then the bytes received to that point are returned from the *read*().

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

33(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is greater than 0, and TIME is greater than 0, then a call to *read*() blocks until the MIN and TIME mechanisms are activated by the first byte received or by a signal being received.

34(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is greater than 0, TIME is greater than 0, and data is in the buffer at the time of the call to *read*(), then the timer is started as if data had been received immediately after the call to *read*().

7.1.1.7.2 CASE B: MIN > 0, TIME = 0

35(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is greater than 0, and TIME is equal to 0, then a call to *read*() blocks until MIN bytes are received or until a signal is received.

7.1.1.7.3 CASE C: MIN = 0, TIME > 0

36(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is equal to 0, and TIME is greater than 0, then a call to *read*() is satisfied when any data is received before TIME × 0,1 s pass.

37(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is equal to 0, TIME is greater than 0, the timer expires after TIME × 0,1 s, and the read queue is empty, then zero bytes are returned from the call to *read*().

38(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is equal to 0, TIME is greater than 0, and data is in the buffer at the time of the call to *read*(), then the timer is started as if data had been received immediately after the call to *read*().

7.1.1.7.4 CASE D: MIN = 0, TIME = 0

39(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is equal to 0, TIME is equal to 0, and there is enough data available to satisfy the request, then a call to *read*() returns the bytes requested.

40(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is equal to 0, TIME is equal. to 0, and there is some data available but less that the amount requested, then a call to *read*() returns the number of bytes currently available.

41(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is clear, MIN is equal to 0, TIME is equal to 0, and there is no data available, then a call to *read*() returns zero bytes.

7.1.1.8 Writing Data and Output Processing

D11(C) If support or nonsupport for the condition when the implementation buffers *write*() output to a terminal device is documented:

The details are contained in 7.1.1.8 of the PCD.1.

42(PCTS_GTI_DEVICE?A:UNTESTED)

When O_NONBLOCK is set by *open*() or *fcntl*(), and when no data can be accepted at the time of the *write*(), then a call to *write*() does not block the process, returns a value of (*ssize_t*)-1, sets errno to [EAGAIN], and transfers no data.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

43(PCTS_GTI_DEVICE?A:UNTESTED)

When O_NONBLOCK is clear, then a call to *write*() is blocked until the data can be accepted or a signal has been received.

44(PCTS_GTI_DEVICE?B:UNTESTED)

When O_NONBLOCK is set and some data can be written, then a call to *write*() writes what it can and returns the number of bytes written.

Testing Requirements:

Test for O_NONBLOCK set by *open*() and *fcntl*().

45(PCTS_GTI_DEVICE?B:UNTESTED)

When a *write*() to a terminal returns successfully: then all of the bytes written have already been scheduled for transmission to the device.

See Reason 1 in Section 5. of POSIX.3 [4].

7.1.1.9 Special Characters

46(PCTS_GTI_DEVICE?A:UNTESTED)

When ISIG is set and INTR is input from a terminal, then a SIGINT signal is sent to all processes in the foreground process group for which the terminal is the controlling terminal.

47(PCTS_GTI_DEVICE?A:UNTESTED)

When ISIG is set, then INTR is not placed on the input queue.

48(PCTS_GTI_DEVICE?A:UNTESTED)

When ISIG is set and the QUIT character is input from a terminal, then a SIGQUIT signal is sent to all processes in the foreground process group for which the terminal is the controlling terminal.

49(PCTS_GTI_DEVICE?A:UNTESTED)

When ISIG is set, then QUIT is not placed on the input queue.

50(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When ISIG is set and the SUSP character is input from a terminal, then a SIGTSTP signal is sent to all processes in the foreground process group for which the terminal is the controlling terminal.

51(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When ISIG is set, then SUSP is not placed on the input queue.

R05 When IXON is set, then STOP and START are not placed on the input queue. (See Assertion 31 in 7.1.2.2.)

52(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is set, and ERASE is received, then the last character of the current line is removed from the input queue, line delimiters are not removed, and the ERASE character is discarded. Lines in the input queue are delimited by an NL or by the receipt, of an EOF or EOL character.

Testing Requirements:

Test when current line is empty and when current line contains data.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

53(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is set, and KILL is input from a terminal, then all characters in the current line are removed from the input queue, up to but not including the line delimiter, and the KILL character is discarded. Lines in the input queue are delimited by a NL or by the receipt of an EOF or EOL character.

54(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is set, EOF is received as input data, and fewer bytes of input data are received prior to an EOF than those requested by the *read*(), then the *read*() returns the input data prior to the EOF, and the EOF character is discarded.

Testing Requirements:

Test for the conditions when the number of bytes requested is N and the bytes waiting to be read are 0 and N -1.

55(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is set, EOF is received as input data, and more bytes of input data are received prior to an EOF than those requested by the *read*(), then the *read*() returns the input data requested, and the remaining bytes and the EOF are retained in the input stream.

Testing Requirements:

Test for the condition when the number of bytes requested is N and the bytes waiting to be read are N + 1.

56(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON is set, EOF is received as input data, and the bytes of input data received prior to an EOF is the number of bytes requested by *read*(), then the *read*() returns the input data requested.

Testing Requirements:

Test for the condition when the number of bytes requested is N and the bytes waiting to be read are N.

NOTE — An IEEE interpretation is needed to determine if the EOF is retained or deleted.

R06 When ICANON is set, and when NL (the C Standard {2} language character constant '\n') is input from a terminal, then a line is delimited. (See Assertion 23 in 7.1.1.6.)

R07 When ICANON is set and EOL is input from a terminal, then a line is delimited. (See Assertion 23 in 7.1.1.6.)

R08 When IXON is set and STOP is input from a terminal, then output to that terminal is suspended. (See Assertion 29 in 7.1.2.2.)

R09 When IXON is set, START is input from a terminal, and output to that terminal is suspended by the receipt of STOP, then output to that terminal is recommenced. (See Assertion 30 in 7.1.2.2.)

57(PCTS_GTI_DEVICE?A:UNTESTED)

The values for INTR, QUIT, ERASE, KILL, EOF, and EOI, characters can be changed.

58(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

The SUSP character can be changed.

D12(A) The details on whether the START and STOP special characters can be changed are contained in 7.1.1.9 of the PCD.1.

59(PCTS_GTI_DEVICE?C:UNTESTED)

If the changing of the START and STOP characters is supported:

The values for the START and STOP characters can be changed.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860340 484 ■

60(PCTS_GTI_DEVICE?A:UNTESTED)

When the IEXTEN flag is not set, then all characters except the special characters INTR, QUIT, SUSP, ERASE, KILL, EOF, EOL, START, and STOP are received without interpretation.

61(PCTS_GTI_DEVICE?C:UNTESTED)

If {_POSIX_VDISABLE} is in effect for the terminal file:

When the value of the special character is set to the value {_POSIX_VDISABLE}, then the action associated with that special character (INTR, QUIT, ERASE, KILL, EOF, or EOL) does not occur for any character received.

62(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported, and {_POSIX_VDISABLE} is in effect for the terminal file:

When the value of SUSP is set to the value {_POSIX_VDISABLE}, then the SUSP action does not occur for any character received.

63(PCTS_GTI_DEVICE?C :UNTESTED)

If {_POSIX_VDISABLE} is in effect for the terminal file, and the START and STOP characters can be changed:

When the values of START and STOP are set to the value {_POSIX_VDISABLE}, then the function of that character does not occur for any character received.

D13(C) If the response of the terminal interface when two or more special characters have the same value is documented:

The details of the operation performed when that character is received are contained in 7.1.1.9 of the PCD.1.

D14(C) If any single bytes other than INTR, QUIT, ERASE, KILL, EOF, NL, EOL, SUSP, STOP, START, or CR, or any multibyte character sequences, have special meaning in terminal input:

The details are contained in 7.1.1.9 of the PCD.1.

7.1.1.10 Modem Disconnect

64(PCTS_GTI_DEVICE?C:UNTESTED)

If modem control is supported:

When a modem disconnect is detected by the terminal interface for a controlling terminal, and when CLOCAL is clear, then the SIGHUP signal is sent to the controlling process associated with the terminal.

65(PCTS_GTI_DEVICE?C:UNTESTED)

If modem control is supported:

When a modem disconnect is detected, and when SIGHUP is blocked, caught, or ignored, then any subsequent *read*() to the disconnected terminal file returns zero, indicating end-of-file. This behavior continues for each subsequent call to *read*() until the device is closed.

D15(C) If both the behavior associated with {_POSIX_JOB_CONTROL} and modem control are supported, and it is documented whether an EOF condition is returned or *errno* is set to [EIO] when a modem disconnect is detected by the terminal interface:

The details are contained in 7.1.1.10 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

66(PCTS_GTI_DEVICE?C:UNTESTED)

If modem control is supported:

When a modem disconnect is detected, then any subsequent *write*() to the disconnected terminal file returns a value of (*ssize_t*)-1 and sets *errno* to [EIO]. This behavior continues for each subsequent call to *write*() until the device is closed.

7.1.1.11 Closing a Terminal Device File

67(PCTS_GTI_DEVICE?A:UNTESTED)

The last process to close a terminal device file causes any data in the output queue to be sent to the device.

68(PCTS_GTI_DEVICE?B:UNTESTED)

The last process to close a terminal device file causes any data in the input queue to be discarded.

See Reason 3 in Section 5. of POSIX.3 [4].

69(PCTS_GTI_DEVICE?C:UNTESTED)

If both modem control and HUPCL set are supported:

When the last close on the device occurs and the HUPCL flag is set, then a disconnect is performed.

7.1.2 Parameters That Can Be Set

7.1.2.1 *termios* Structure

01(A) When <termios.h> is included, then *struct terrnios* is declared.

D01(C) If the *termios* structure of the implementation includes members other than those specified by POSIX.1 {3}, and this is documented:

The details are contained in 7.1.2.1 of the PCD.1.

02(A) When <termios.h> is included, then the types *tcflag_t* and *cc_t* are defined as unsigned integral types.

03(A) When <terminos.h> is included, then the elements of *struct termios*: *c_iflag*, *c_oflag*, *c_cflag*, and *c_lflag* are defined, each element being of type *tcflag_t*.

04(A) When <termios.h> is included, then NCCS is defined as a positive integral constant.

R01 NCCS is greater than the largest special control character subscript specifically defined in POSIX.1 {3}. (See Assertion 70 in 7.1.2.6.)

05(B) NCCS is greater than or equal to the number of distinct special control character subscripts available to an implementation.

See Reason 4 in Section 5. of POSIX.3 [4].

06(A) When <termios.h> is included, then *c_cc* is defined as an array having NCCS elements of type *cc_t*.

7.1.2.2 Input Modes

07(A) When <termios.h> is included, then the symbolic masks IGNBRK, BRKINT, IGNPAR, PARMRK, INPCK, ISTRIP, INLCR, IGNCR, ICRNL, IXON, and IXOFF are defined and are bitwise distinct from each other.

D02(A) The details of the break condition for non-asynchronous data transmission are contained in 7.1.2.2 of the PCD.1.

08(PCTS_GTI_DEVICE?A:UNTESTED)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When IGNBRK is set, then a break condition detected on input is ignored and does not affect the data read by any process.

09(PCTS_GTI_DEVICE?A:UNTESTED)

When IGNBRK is clear and BRKINT is set, then the break condition flushes the input and output queues associated with that terminal.

Testing Requirements:

Test for a controlling terminal and a noncontrolling terminal.

NOTE — When the implementation does not buffer output for the terminal, then testing for flushing of output queues does not apply.

10(PCTS_GTI_DEVICE?A:UNTESTED)

When IGNBRK is clear, BRKINT is set, and the terminal is the controlling terminal of a foreground process group, then the break condition generates a single SIGINT signal to the foreground process group.

11(PCTS_GTI_DEVICE?A:UNTESTED)

When IGNBRK and BRKINT and PARMRK are clear, then a break condition is read as a single '\0'.

12(PCTS_GTI_DEVICE?A:UNTESTED)

When IGNBRK and BRKINT are clear, and PARMRK is set, a break condition is read as '\377', '\0', '\0'.

Testing Requirements:

Test with ISTRIP set and with ISTRIP clear.

13(PCTS_GTI_DEVICE?B:UNTESTED)

When IGNPAR is set, then a byte with a framing error (other than break) is ignored.

See Reason 1 in Section 5. of POSIX.3 [4].

14(PCTS_GTI_DEVICE?C:UNTESTED)

If PARENB set is supported:

When PARENB, INPCK, and IGNPAR are set, then a byte with a parity error is ignored.

15(PCTS_GTI_DEVICE?B:UNTESTED)

When PARMRK is set, and IGNPAR is clear, then a byte with a framing error (other than break) is read as the three-character sequence '\377', '\0', 'X', where 'X' is the data of the character received in error.

Testing Requirements:

Test with ISTRIP set and with ISTRIP clear.

See Reason 1 in Section 5. of POSIX.3 [4].

16(PCTS_GTI_DEVICE?C:UNTESTED)

If PARENB set is supported:

When PARENB, INPCK, and PARMRK are set and IGNPAR is clear, then a byte with a parity error is read as the three-character sequence '\377', '\0', 'X', where 'X' is the data of the character received in error.

Testing Requirements:

Test with ISTRIP set and with ISTRIP clear.

17(PCTS GTI_DEVICE?C :UNTESTED)

If both PARENB set and a CSIZE of CS8 are supported:

When PARENB and PARMRK are set, when CSIZE is set to CS8, and when IGNPAR and ISTRIP are clear, then a valid character of '\377' is read as '\377', '\377'.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

18(PCTS_GTI_DEVICE?B:UNTESTED)

When IGNPAR and PARMRK are clear, then a framing error (other than break) is read as single character '\0'.

See Reason 1 in Section 5. of POSIX.3 [4].

19(PCTS_GTI_DEVICE?C:UNTESTED)

If PARENB set is supported:

When PARENB and INPCK are set, and IGNPAR and PARMRK are clear, then a parity error is given to the application as a single character '\0'.

20(PCTS_GTI_DEVICE?C:UNTESTED)

If PARENB set is supported:

When PARENB and INPCK are set, then input parity checking is enabled, and characters received with correct parity are as sent and do not generate an error.

21(PCTS_GTI_DEVICE?C:UNTESTED)

If PARENB set is supported:

When PARENB is set and INPCK is clear, then input parity checking is disabled but output parity generation will occur.

22(PCTS_GTI_DEVICE?C:UNTESTED)

If a CSIZE of CS8 is supported:

When CSIZE is set to CS8 and ISTRIP is set, then valid input characters are stripped to 7 least significant bits before returning them to the application.

23(PCTS_GTI_DEVICE?C:UNTESTED)

If a CSIZE of CS8 is supported:

When CSIZE is set to CS8 and ISTRIP is clear, then all 8 b of valid input characters are returned to the application.

24(PCTS_GTI_DEVICE?A:UNTESTED)

When INLCR is set, then a received NL character is translated into a CR character.

25(PCTS_GTI_DEVICE?A:UNTESTED)

When INLCR is clear, then a received NL character is placed on the input queue unaltered.

26(PCTS_GTI_DEVICE?A:UNTESTED)

When IGNCR is set, then a received CR character is not placed in the input queue.

27(PCTS_GTI_DEVICE?A:UNTESTED)

When IGNCR is clear and ICRNL is set, then a received CR character is translated into a NL character.

28(PCTS_GTI_DEVICE?A:UNTESTED)

When IGNCR and ICRNL are both clear, then a received CR character is not altered during input processing.

29(PCTS_GTI_DEVICE?A:UNTESTED)

When IXON is set, then a received STOP character suspends output to the terminal.

30(PCTS_GTI_DEVICE?A:UNTESTED)

When IXON is set, then a received START character restarts output to the terminal suspended by a previous STOP character.

31(PCTS_GTI_DEVICE?A:UNTESTED)

When IXON is set, then no START or STOP characters are placed in the input queue.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

32(PCTS_GTI_DEVICE?A:UNTESTED)

When IXON is clear, then the START and STOP characters are placed in the input queue.

33(PCTS_GTI_DEVICE && {MAX_INPUT} ≤ {PCTS_MAX_INPUT}?A:UNTESTED)

When IXOFF is set, then the system transmits one or more STOP characters intended to prevent the input queue from overflowing.

34(PCTS_GTI_DEVICE && {MAX_INPUT} ≤ {PCTS_MAX_INPUT}?A:UNTESTED)

When IXOFF is set, when the system has stopped input from the terminal device by sending one or more STOP characters, and when sufficient data has been read from the input queue to allow input to be restarted, then the system transmits one or more START characters as needed to cause the terminal device to resume transmitting data.

D03(A) The details of the precise conditions under which STOP and START characters are transmitted to prevent the number of bytes in the input queue from exceeding {MAX_INPUT} are contained in 7.1.2.2 of the PCD.1.

D04(A) The initial input control value after *open*() is specified in 7.1.2.2 of the PCD.1.

7.1.2.3 Output Modes

35(A) When <termios.h> is included, then the mask OPOST is defined.

D05(A) The details describing the manner in which output data is processed when OPOST is set to modify the appearance of lines of text on the terminal device are contained in 7.1.2.3 of the PCD.1.

36(PCTS_GTI_DEVICE?A:UNTESTED)

When OPOST is clear, then characters are transmitted to the terminal without change.

D06(A) The initial output control value after *open*() is specified in 7.1.2.3 of the PCD.1.

7.1.2.4 Control Modes

D07(C) If hardware control modes are ignored for an object that is not an asynchronous serial connection, and this is documented:

The details are contained in 7.1.2.4 of the PCD.1.

37(A) When <termios.h> is included, then the symbolic flags CLOCAL, CREAD, CSIZE, CSTOPB, HUPCL, PARENB, and PARODI) are defined and are bitwise distinct from each other.

38(A) When <termios.h> is included, then the CS5, CS6, CS7, and CS8 bit masks are defined to have distinct values. CSIZE is defined as a mask that includes all bits in the bitwise inclusive OR of those values.

39(PCTS_GTI_DEVICE?C:UNTESTED)

If CSIZE of CS5 is supported:

When CSIZE bits are set to CS5, then characters are sent and received as 5 b, not including the parity bit.

40(PCTS_GTI_DEVICE?C:UNTESTED)

If a CSIZE of CS6 is supported:

When CSIZE bits are set to CS6, then characters are sent and received as 6 b, not including the parity bit.

41(PCTS_GTI_DEVICE?C:UNTESTED)

If a CSIZE of CS7 is supported:

When CSIZE bits are set to CS7, then characters are sent and received as 7 b, not including the parity bit.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

42(PCTS_GTI_DEVICE?C:UNTESTED)

If a CSIZE of CS8 is supported:
When CSIZE bits are set to CS8, then characters are sent and received as 8 b, not including the parity bit.

43(PCTS_GTI_DEVICE?D:UNTESTED)

If CSTOPB set is supported:
When CSTOPB is set, then two stop bits are used.

See Reason 1 in Section 5. of POSIX.3 [4].

44(PCTS_GTI_DEVICE?D:UNTESTED)

If CSTOPB clear is supported:
When CSTOPB is clear, then one stop bit is used.

See Reason 1 in Section 5. of POSIX.3 [4].

45(PCTS_GTI_DEVICE?C:UNTESTED)

If CSTOPB set is supported:
When CSTOPB is set, then data can be written and read.

46(PCTS_GTI_DEVICE?C:UNTESTED)

If CSTOPB clear is supported:
When CSTOPB is clear, then data can be written and read.

47(PCTS_GTI_DEVICE?C:UNTESTED)

If CREAD set is supported:
When CREAD is set, then characters can be received.

48(PCTS_GTI_DEVICE?C:UNTESTED)

If CREAD clear is supported:
When CREAD is clear, then characters cannot be received.

R02 When PARENB is set, then parity generation and detection is enabled. (See Assertions 49-52 in 7.1.2.4.)

49(PCTS_GTI_DEVICE?D:UNTESTED)

If both PARENB set and PARODD set are supported:
When PARODD is set, then parity generation uses odd parity.

See Reason 1 in Section 5. of POSIX.3 [4].

50(PCTS_GTI_DEVICE?C:UNTESTED)

If both PARENB set and PARODD set are supported:
When PARENB and PARODD are set, then parity generation and detection is enabled.
Testing Requirements:
Test that characters received with odd parity do not generate an error and that characters received with even parity generate an error.

51(PCTS_GTI_DEVICE?D:UNTESTED)

If both PARENB set and PARODD clear are supported:
When PARODD is clear, then parity generation uses even parity.

See Reason 1 in Section 5. of POSIX.3 [4].

52(PCTS_GTI_DEVICE?C:UNTESTED)

If both PARENB set and PARODD clear are supported:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When PARENB is set and PARODD is clear, then parity generation and detection is enabled.
Testing Requirements:
Test that characters received with even parity do not generate an error and that characters received with odd parity generate an error.

53(PCTS_GTI_DEVICE?C:UNTESTED)

If both modem control and HUPCL set are supported:
When HUPCL is set, CLOCAL is clear, and the last process with the port open closes the port or terminates, then modem disconnect occurs.

54(PCTS_GTI_DEVICE?C:UNTESTED)

If both modem control and HUPCL clear are supported:
When both HUPCL and CLOCAL are clear, and the last process with the port open closes the port or terminates, then modem disconnect does not occur.

55(PCTS_GTI_DEVICE?C:UNTESTED)

If CLOCAL set is supported:
When CLOCAL is set, then a connection is independent of the state of the modem status lines. The SIGHUP signal is not sent to the controlling process when the modem control line is lowered.

R03 When CLOCAL is clear, then a connection is dependent on the state of the modem status lines. The SIGHUP signal is sent to the controlling process when a modem disconnect is detected. (See Assertion 64 in 7.1.1.10.)

56(PCTS_GTI_DEVICE?C:UNTESTED)

For *funct*() of *open*() and *creat*():
If modem control is supported:
When CLOCAL is clear for the terminal, then a call to *funct*()—with the O_NONBLOCK flag clear in the case where *funct*() is *open*()—waits for the modem connection tc complete before returning.

57(PCTS_GTI_DEVICE?C:UNTESTED)

If modem control is supported:
When CLOCAL is clear for the terminal, then a call to *open*() with the O_NONBLOCK flag set returns without waiting for the modem connection.

58(PCTS_GTI_DEVICE?C:UNTESTED)

For *funct*() of *open*() and *creat*():
If CLOCAL set is supported:
When CLOCAL is set for the terminal, then a call to *funct*() returns without waiting for the modem connection.
Testing Requirements:
Test *open*() with O_NONBLOCK set and O_NONBLOCK clear.

D08(A) The initial hardware control value after *open*() is specified in 7.1.2.4 of the PCD.1.

7.1.2.5 Local Modes

59(A) When `<termios.h>` is included, then the symbols ECHO, ECHOE, ECHOK, ECHONL, ICANON, IEXTEN, ISIG, NOFLSH, and TOSTOP are defined and are bitwise distinct from each other.

60(PCTS_GTI_DEVICE?A:UNTESTED)

When ECHO is set, then input characters are echoed back to the terminal.

61(PCTS_GTI_DEVICE?A:UNTESTED)

When ECHO is clear, then input characters are not echoed back to the terminal.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D09(C) If the condition when ECHOE and ICANON are set and an ERASE character is encountered and there is no character to erase, and this is documented:

The details on whether the implementation echos an indication that there was no character to erase or does nothing are contained in 7.1.2.5 of the PCD.1.

62(PCTS_GTI_DEVICE?A: UNTESTED)

When ICANON and ECHOE are set, when there are characters on the current line, and when the ERASE character is received, then the {PCTS_ECHOE} string is sent to the terminal.

Testing Requirements:

Test only for ECHO set.

The PCTS.1 must state the character sequences that are expected to be erased and allow the implementation to provide different sequences for PCTS_ECHOE and PCTS_ECHOK to erase the character sequences for the default terminal type.

D10(C) If the behavior of the terminal interface when ECHOK and ICANON are set is documented:

The details as to whether the terminal causes the terminal to erase the line from the display or echoes the KILL character followed by the '\n' character are contained in 7.1.2.5 of the PCD.1.

63(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON and ECHOK are set and the KILL character is received, then the {PCTS_ECHOK} string is sent to the terminal.

Testing Requirements:

Test only for ECHO set.

The PCTS.1 must state the character sequences that are expected to be erased and allow the implementation to provide different sequences for PCTS_ECHOE and PCTS_ECHOK to erase the character sequences for the default terminal type.

64(PCTS_GTI_DEVICE?A:UNTESTED)

When ICANON and ECHONL are set, then the NL character is echoed when received.

Testing Requirements:

Test with ECHO set and with ECHO clear.

65(PCTS_GTI_DEVICE?A:UNTESTED)

When ISIG is clear, and INTR or QUIT is received, then no special character functions occur, and the characters are placed on the input queue.

66(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When ISIG is clear and SUSP is received, then no special character function occurs, and the SUSP character is placed on the input queue.

R04 When ICANON is set, then canonical mode input processing is enabled. (See Assertions 23-26 in 7.1.1.6.)

R05 When ICANON is clear, then read requests are satisfied directly from the input queue in accordance with noncanonical mode input processing. (See Assertions 27 and 28 in 7.1.1.7.)

D11(C) If any functions are recognized from input data when IEXTEN is set:

These functions are stated in 7.1.2.5 of the PCD.1.

D12(A) The interaction of IEXTEN being set with ICANON, ISIG, IXON, or IXOFF is described in 7.1.2.5 of the PCD.1.

67(PCTS_GTI_DEVICE?A:UNTESTED)

When NOFLSH and ISIG are set, and INTR or QUIT is received, then data in the input and output queues is not discarded.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NOTE — INTR and QUIT are discarded when processed.

68(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When NOFLSH and ISIG are set, and SUSP is received, then data in the input and output queues is not discarded.
NOTE — SUSP is discarded when processed.

R06 When TOSTOP is set, then TOSTOP signalling is enabled for a process writing to the controlling terminal. (See Assertions 16–18 in 7.1.1.4.)

D13(A) The initial local control value after *open*() is specified in 7.1.2.5 of the PCD.1.

7.1.2.6 Special Control Characters

69(A) When `<termios.h>` is included, then the subscripts VEOF, VEOL, VERASE, VINTR, VKILL, VMIN, VQUIT, VSUSP, VTIME, VSTART, and VSTOP are defined.

70(A) VERASE, VINTR, VKILL, VQUIT, VSUSP, VSTART, and VSTOP are all distinct and less than NCCS. VMIN and VEOF are distinct from all of the above (but not necessarily from each other) and are less than NCCS. VTIME and VEOL are distinct from all of the above (including VMIN and VEOF) (but not necessarily from each other) and are less than NCCS.

D14(C) If the implementation does not support the behavior associated with {_POSIX_JOB_CONTROL} and it is documented whether the implementation ignores the character value in the *c_cc* array indexed by the VSUSP subscript:
The details are contained in 7.1.2.6 of the PCD.1.

D15(C) If the value of NCCS (the number of elements in the *c_cc* array is documented:
The details are contained in 7.1.2.6 of the PCD.1.

D16(C) If the implementation does not support changing the START and STOP characters and it is documented whether the character values in the *c_cc* array indexed by START and STOP are ignored:
The details are contained in 7.1.2.6 of the PCD.1.

71(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *tcgetattr*() returns the values for START and STOP in the appropriate elements of the *c_cc* array of the *struct termios*.

R07 The function of the special characters INTR, QUIT, ERASE, KILL, EOF, EOL, and SUSP are disabled individually by setting the value of the special character to {_POSIX_VDISABLE}. (See Assertions 61 and 62 in 7.1.1.9.)

D17(A) The initial values of all control characters are stated in 7.1.2.6 of the PCD.1.

7.1.2.7 Baud Rate Values

There are no assertions specific to this subclause.

7.1.3 Baud Rate Functions

7.1.3.1 *cfgetospeed*() (7.1.3)

7.1.3.1.1 Synopsis (7.1.3.1)

01(A) If the implementation provides C Standard {2} support:
When the header `<termios.h>` is included, then the function prototype
`speed_t cfgetospeed(const struct termios *)` is declared. (See GA36 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Otherwise:

When the header <termios.h> is included, then the function *cfgetospeed*() is declared with the result type *speed_t*. (See GA36 in 2.7.3.)

02(C) If *cfgetospeed*() is defined as a macro when the header <termios.h> is included:
When the macro cfgetospeed() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *speed_t*. (See GA37 in 2.7.3.)

03(C) If *cfgetospeed*() is defined as a macro in the header <termios.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

7.1.3.1.2 Description (7.1.3.2)

04(A) When <termios.h> is included, then the symbolic baud rates B0, B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200, and B38400 are defined.

05(A) The type *speed_t* is defined in <termios.h> and is an unsigned integral type.

06(A) A call to *cfgetospeed(termios_p)* returns the output baud rate stored in the *termios* structure pointed to by the *termios_p* parameter.

7.1.3.1.3 Returns (7.1.3.3)

There are no assertions specific to this subclause.

7.1.3.1.4 Errors (7.1.3.4)

D01(C) If error conditions are detected for *cfgetospeed*(), and this is documented:
The details on the error conditions detected are contained in 7.1.3.4 of the PCD.1.

7.1.3.2 *cfsetospeed*() (7.1.3)

7.1.3.2.1 Synopsis (7.1.3.1)

01(A) If the implementation provides C Standard {2} support:
When the header <termios.h> is included, then the function prototype
`int cfsetospeed(struct termios *, speed_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <termios.h> is included, then the function *cfsetospeed*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *cfsetospeed*() is defined as a macro when the header <termios.h> is included:
When the macro *cfsetospeed*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type int. (See GA37 in 2.7.3.)

03(C) If *cfsetospeed*() is defined as a macro in the header <termios.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

7.1.3.2.2 Description (7.1.3.2)

04(A) A call to *cfsetospeed(termios_p, speed)* sets the output baud rate stored in the *termios* structure pointed to by the *termios_p* parameter to *speed*.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

05(PCTS_GTI_DEVICE?A:UNTESTED)

For each of the following baud rates that are supported: B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200, and B38400, a call to *cfsetospeed*() that returns 0, followed by a *tcsetattr*() that returns 0, sets the output baud rate to the corresponding speed. At least one baud rate must be supported.

D01(C) If it is documented whether *cfsetospeed*() returns an error when an attempt is made to set an unsupported baud rate:

The details are contained in 7.1.3.2 and 7.1.3.4 of the PCD.1.

7.1.3.2.3 Returns (7.1.3.3)

There are no assertions specific to this subclause.

7.1.3.2.4 Errors (7.1.3.4)

D02(C) If error conditions are detected for *cfsetospeed*(), and this is documented:

The details on the error conditions detected are contained in 7.1.3.4 of the PCD.1.

7.1.3.3 *cfgetispeed*() (7.1.3)

7.1.3.3.1 Synopsis (7.1.3.1)

01(A) If the implementation provides C Standard {2} support:

When the header <termios.h> is included, then the function prototype `speed_t cfgetispeed(const struct termios *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <termios.h> is included, then the function *cfgetispeed*() is declared with the result type *speed_t*. (See GA36 in 2.7.3.)

02(C) If *cfgetispeed*() is defined as a macro when the header <termios.h> is included:

When the macro *cfgetispeed*() is invoked with the correct argument types (or compatible argument types in the case, that C Standard {2} support is provided), then it expands to an expression with the result type *speed_t*. (See GA37 in 2.7.3.)

03(C) If *cfgetispeed*() is defined as a macro in the header <termios.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

7.1.3.3.2 Description (7.1.3.2)

04(A) A call to *cfgetispeed*(*termios_p*) returns the input baud rate stored in the termios structure pointed to by the *termios_p* parameter.

7.1.3.3.3 Returns (7.1.3.3)

There are no assertions specific to this subclause.

7.1.3.3.4 Errors (7.1.3.4)

D01(C) If error conditions are detected for *cfgetispeed*(), and this is documented:

The details on the error conditions detected are contained in 7.1.3.4 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.1.3.4 *cfsetispeed*() (7.1.3)

7.1.3.4.1 Synopsis (7.1.3.1)

01(A) If the implementation provides C Standard {2} support:
When the header <termios.h> is included, then the function prototype
`int cfsetispeed(struct termios *, speed_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <termios.h> is included, then the function *cfsetispeed*() is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *cfsetispeed*() is defined as a macro when the header <termios.h> is included:
When the macro *cfsetispeed*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *cfsetispeed*() is defined as a macro in the header <termios.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

7.1.3.4.2 Description (7.1.3.2)

04(A) A call to *cfsetispeed*(*termios_p*, *speed*) sets the input baud rate stored in the *termios* structure pointed to by the *termios_p* parameter to *speed*.

05(PCTS_GTI_DEVICE?A:UNTESTED)

For each of the following baud rates that are supported: B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800 B2400, B4800, B9600, B19200, and B38400, a call to *cfsetispeed*() that returns 0, followed by a call to *tcsetattr*() that returns 0, sets the input baud rate to the corresponding speed. At least one baud rate must be supported.

D01(C) If it is documented whether *cfsetispeed*() returns an error when an attempt is made to set an unsupported baud rate:
The details are contained in 7.1.3.2 and 7.1.3.4 of the PCD.1.

7.1.3.4.3 Returns (7.1.3.3)

There are no assertions specific to this subclause.

7.1.3.4.4 Errors (7.1.3.4)

D02(C) If error conditions are detected for *cfsetispeed*(), and this is documented:
The details on the error conditions detected are contained in 7.1.3.4 of the PCD.1.

7.2 General Terminal Interface Control Functions

GA60 For *funct*() of *tcsetattr*(), *tcsendbreak*(), *tcdrain*(), *tcflow*(), and *tcflush*():
If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
A call to *funct*() on the controlling terminal of the process from a member of a background process group that is not blocking or ignoring SIGTTOU signals causes a SIGTTOU signal to be sent to its process group. The operation of *funct*()is not performed prior to the signal being received.
Testing Requirements:
Test when TOSTOP is set and when TOSTOP is clear.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

GA61 For *funct*() of *tcsetattr*(), *tcsendbreak*(), *tcdrain*(), *tcflow*(), and *tcflush*():
If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a member of a background process group that is blocking SIGTTOU signals calls *funct*() on the controlling terminal of the process, then *funct*() is allowed to perform the actions and returns the value zero, and the SIGTTOU signal is not sent to its process group.
Testing Requirements:
Test when TOSTOP is set and when TOSTOP is clear.

GA62 For *funct*() of *tcsetattr*(), *tcsendbreak*(), *tcdrain*(), *tcflow*(), and *tcflush*():
If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a member of a background process group that is ignoring SIGTTOU signals calls *funct*() on the controlling terminal of the process, then *funct*() is allowed to perform the actions and returns the value zero, and the SIGTTOU signal is not sent to its process group.
Testing Requirements:
Test when TOSTOP is set and when TOSTOP is clear.

There are no assertions specific to this subclause.

7.2.1 Get and Set State

7.2.1.1 *tcgetattr*() (7.2.1)

7.2.1.1.1 Synopsis (7.2.1.1)

01(A) If the implementation provides C Standard {2} support:
When the header <termios.h> is included, then the function prototype
`int tcgetattr(int, struct termios *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <termios.h> is included, then the function *tcgetattr*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *tcgetattr*() is defined as a macro when the header <termios.h> is included:
When the macro *tcgetattr*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *tcgetattr*() is defined as a macro in the header <termios.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

7.2.1.1.2 Description (7.2.1.2)

04(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *tcgetattr(fildes, termios_p)* stores the parameters associated with the terminal in the termios structure referenced by *termios_p* and returns zero.

Testing Requirements:
If {_POSIX_JOB_CONTROL} is supported, test the behavior for a process that is a member of a foreground process group and for a process that is a member of a background process group.

D01(C) If support or nonsupport for input and output baud rates that differ is documented:
The details are contained in 7.2.1.2 of the PCD.1.

05(PCTS_GTI_DEVICE?C:UNTESTED)

If the implementation supports input and output baud rates that differ:

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The baud rates stored in the *termios* structure returned by *tcgetattr*() reflect the actual baud rates even when they are equal.

06(PCTS_GTI_DEVICE?C:UNTESTED)

If the implementation does not support input and output baud rates that differ:

The output baud rate stored in the *termios* structure returned by *tcgetattr*() is the actual baud rate. The input baud rate stored in this *termios* structure is either equal to zero or equal to the actual baud rate.

7.2.1.1.3 Returns (7.2.1.3)

R01 When a call to *tcgetattr*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 4 in 7.2.1.1.2.)

R02 When a call to *tcgetattr*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error. (See Assertions 7 and 8 in 7.2.1.1.4.)

7.2.1.1.4 Errors (7.2.1.4)

07(A) When *fildes* is not a valid file descriptor, then a call to *tcgetattr(fildes, termios_p)* returns a value of (*int*)−1 and sets *errno* to [EBADF].

08(A) When the file associated with *fildes* is not a terminal, then a call to *tcgetattr(fildes, termios_p)* returns a value of (*int*)−1 and sets *errno* to [ENOTTY].

7.2.1.2 *tcsetattr*() (7.2.1)

7.2.1.2.1 Synopsis (7.2.1.1)

01(A) If the implementation provides C Standard {2} support:

When the header <termios.h> is included, then the function prototype `int tcsetattr(int, int, const struct termios *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <termios.h> is included, then the function *tcsetattr*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *tcsetattr*() is defined as a macro when the header <termios.h> is included:

When the macro tcsetattr() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *tcsetattr*() is defined as a macro in the header <termios.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

7.2.1.2.2 Description (7.2.1.2)

04(A) When the header <termios.h> is included, then the symbols TCSANOW, TCSADRAIN, and TCSAFLUSH are defined and are different from each other.

05(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *tcsetattr(fildes, TCSANOW, termios_p)* from a foreground process immediately sets the parameters associated with the terminal from the *termios* structure referenced by *termios_p*, without waiting for data to drain or to be flushed, and returns the value zero.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

06(PCTS_GTI_DEVICE?B:UNTESTED)

A call to *tcsetattr(fildes, optional_action, termios_p)* from a foreground process sets the parameters associated with the terminal from the *termios* structure referenced by *termios_p* after the specified *optional_action* has been performed.

Testing Requirements:

Test for *optional_action* of TCSADRAIN and TCSAFLUSH.

See Reason 2 in Section 5. of POSIX.3 {4}.

07(PCTS_GTI_DEVICE?A:UNTESTED)

When a call to *tcsetattr(fildes, TCSADRAIN, termios_p)* is made from a foreground process, then the process outputs all data written to the terminal before returning the value zero.

08(PCTS_GTI_DEVICE?A:UNTESTED)

When a call to *tcsetattr(fildes, TCSAFLUSH, termios_p)* is made from a foreground process, then the process outputs all data written to the terminal and discards all input that has been received but not read before returning the value zero.

09(PCTS_GTI_DEVICE?B:UNTESTED)

A call to *tcsetattr(fildes, TCSAFLUSH, termios_p)* from a foreground process sets the parameters associated with the terminal from the *termios* structure referenced by *termios_p* after all output written to the terminal has been transmitted and all input that has been received but not read is discarded, and returns the value zero.

See Reason 2 in Section 5. of POSIX.3 {4}.

10(PCTS_GTI_DEVICE?C:UNTESTED)

If modem control is supported:

When the output speed in the *termios* structure is set to B0 and *tcsetattr*() is called, then the modem control lines are no longer asserted.

11(PCTS_GTI_DEVICE?A:UNTESTED)

When the input baud rate in the *termios* structure is set to zero and *tcsetattr*() is called, then the input baud rate is changed to the value of the output baud rate as specified in the *termios* structure.

12(PCTS_GTI_DEVICE?C:UNTESTED)

If the implementation does not support an attribute represented in the *termios* structure:

When an attempt is made to change an attribute represented in the *termios* structure to an unsupported value, and when another attribute is changed to a supported value, then a call to *tcsetattr*() returns zero, sets all attributes that the implementation supports as requested, and leaves unchanged any attributes that are not supported.

13(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

A call to *tcsetattr*(), on the controlling terminal of the process, from a member of a background process group that is not blocking or ignoring SIGTTOU signals causes a SIGTTOU signal to be sent to its process group. The terminal attributes are left unchanged, and any flush or drain action specified by *optional_actions* is not performed. (See GA60 in 7.2.)

14(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When a member of a background process group that is blocking SIGTTOU signals calls *tcsetattr*() on the controlling terminal of the process, then *tcsetattr*() is allowed to perform the actions and returns the value zero, and the SIGTTOU signal is not sent to its process group. (See GA61 in 7.2.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When a member of a background process group that is ignoring SIGTTOU signals calls *tcsetattr*() on the controlling terminal of the process, then *tcsetattr*() is allowed to perform the actions and returns the value zero, and the SIGTTOU signal is not sent to its process group. (See GA62 in 7.2.)

16(PCTS_GTI_DEVICE?A:UNTESTED)

When fildes refers to a noncontrolling terminal, then a call to *tcsetattr(fildes, optional_actions, termios_p)* sets the parameters associated with the terminal from the *termios* structure referenced by *termios_p*, carries out the actions indicated by *optional_actions*, and returns a value of zero.

Testing Requirements:

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

Test from a foreground process, a background process that is neither blocking nor ignoring SIGTTOU, a background process that is blocking SIGTTOU, a background process that is ignoring SIGTTOU, and a process that is a member of an orphaned background process group and that is neither blocking nor ignoring SIGTTOU.

7.2.1.2.3 Returns (7.2.1.3)

R01 When a call to *tcsetattr*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 5, 6, 11, 12, and 16 in 7.2.1.2.2.)

R02 When a call to *tcsetattr*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error. (See Assertions 17-21 in 7.2.1.2.4.)

7.2.1.2.4 Errors (7.2.1.4)

17(PCTS_GTI_DEVICE?A:UNTESTED)

When *fildes* is not a valid file descriptor, then a call to *tcsetattr(fildes, optional_actions, termios_p)* returns a value of (*int*)−1 and sets *errno* to [EBADF]. The values in the *termios* structure are not changed.

18(PCTS_GTI_DEVICE && {_EPOSIX_JOB_CONTROL}?A:UNTESTED)

When a call to *tcsetattr*() is interrupted by a signal, then it returns a value of [(*int*)-1] and *errno* is set to [EINTR].

Testing Requirements:

When the behavior associated with {_POSIX_JOB_CONTROL} is supported, test by calling *tcsetattr*() from a member of a background process group that is catching and not blocking SIGTTOU signals. Also test that the values in the termios structure are not changed, the terminal attributes are left unchanged, and any flush or drain action specified by *optional_actions* is not performed.

See Reason 1 in Section 5. of POSIX.3 {4}.

19(PCTS_GTI_DEVICE?A:UNTESTED)

When *optional_actions* has an invalid value, then a call to *tcsetattr(fildes, optional_actions, termios_p)* returns a value of (*int*)-1 and sets *errno* to [EINVAL]. The values in the *termios* structure are not changed, and the terminal atributes are left unchanged.

20(PCTS_GTI_DEVICE?C:UNTESTED)

If the implementation does not support an attribute represented in the *termios* structure:

When an attempt is made to change an attribute represented in the termios structure to an unsupported value, and when no other attribute is changed to a supported value, then a call to *tcsetattr(fildes, optional_actions, termios_p)* returns a value of (*int*)-1 and sets *errno* to [EINVAL].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The values in the *termios* structure and the terminal attributes are left unchanged, and any flush or drain action specified by *optional_actions* is not performed.

21(PCTS_GTI_DEVICE?A:UNTESTED)

When the file associated with *fildes* is not a terminal, then a call *tcsetattr(fildes, optional_actions, termios_p)* returns a value of (*int*)-1 and sets *errno* to [ENOTTY]. The values in the *termios* structure are not changed.

7.2.2 Line Control Functions

7.2.2.1 *tcsendbreak* () (7.2.2)

7.2.2.1.1 Synopsis (7.2.2.1)

01(A) If the implementation provides C Standard {2} support:

When the header <termios.h> is included, then the function prototype `int tcsendbreak(int, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <termios.h> is included, then the function *tcsendbreak*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *tcsendbreak*() is defined as a macro when the header <termios.h> is included:

When the macro *tcsendbreak*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *tcsendbreak*() is defined as a macro in the header <termios.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

7.2.2.1.2 Description (7.2.2.2)

04(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *tcsendbreak(fildes, 0)*, on the controlling terminal of the process, causes transmission of a continuous stream of zero-valued bits for at least 0,25 s, and not more than 0,5 s, and returns the value zero.

D01(A) The period of time the break signal is sent when *tcsendbreak*() is called with a nonzero value for *duration* is specified in 7.2.2.2 of the PCD.1.

D02(C) If non-asynchronous serial data transmission with the terminal is supported:

The details on whether the *tcsendbreak*() function sends data to generate a break condition or returns without any action are contained in 7.2.2.2 of the PCD.1.

05(PCTS_GTI_DEVICE?B:UNTESTED)

A call to *tcsendbreak(fildes, duration)* sends a continuous stream of zero-valued bits for an implementation-defined period of time, and the return value is zero.

See Reason 2 in Section 5. of POSIX.3 {4}.

06(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

A call to *tcsendbreak*(), on the controlling terminal of the process, from a member of a background process group that is not blocking or ignoring SIGTTOU signals causes a SIGTTOU signal to be sent to its process group. The transmission of zero-valued bits is not performed. (See GA60 in 7.2.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

07(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When a member of a background process group that is blocking SIGTTOU signals calls *tcsendbreak*() on the controlling terminal of the process, the *tcsendbreak*() is allowed to perform the actions and returns the value zero, and the SIGTTOU signal is not sent to its process group. (See GA61 in 7.2.)

08(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When a member of a background process group that is ignoring SIGTTOU signals calls *tcsendbreak*() on the controlling terminal of the process, then *tcsendbreak*() is allowed to perform the actions and returns the value zero, and the SIGTTOU signal is not sent to its process group. (See GA62 in 7.2.)

09(PCTS_GTI_DEVICE?A:UNTESTED)

When *fildes* refers to a noncontrolling terminal, then a call to *tcsendbreak(fildes, O)* causes transmission of a break signal and returns the value zero.

Testing Requirements:

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

Test from a foreground process, a background process that is neither blocking nor ignoring SIGTTOU, a background process that is blocking SIGTTOU, a background process that is ignoring SIGTTOU, and a process that is a member of an orphaned background process group and that is neither blocking nor ignoring SIGTTOU.

7.2.2.1.3 Returns (7.2.2.3)

R01 When a call to *tcsendbreak*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 4, 5, 7, and 8 in 7.2.2.1.2.)

R02 When a call to *tcsendbreak*() completes unsuccessfully, then a value of (*int*)−1 is returned and sets *errno* to indicate the error. (See Assertions 10 and 11 in 7.2.2.1.4.)

7.2.2.1.4 Errors (7.2.2.4)

10(A) When *fildes* is not a valid file descriptor, then a call to *tcsendbreak(fildes, duration)* returns a value of (*int*)−1 and sets *errno* to [EBADF].

11(A) When the file associated with *fildes* is not a terminal, then a call to *tcsendbreak(fildes, duration)* returns a value of (*int*)−1 and sets *errno* to [ENOTTY].

7.2.2.2 *tcdrain*() (7.2.2)

7.2.2.2.1 Synopsis (7.2.2.1)

01(A) If the implementation provides C Standard {2} support:

When the header `<termios.h>` is included, then the function prototype `int tcdrain(int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header `<termios.h>` is included, then the function *tcdrain*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *tcdrain*() is defined as a macro when the header `<termios.h>` is included:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the macro *tcdrain*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *tcdrain*() is defined as a macro in the header <termios.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4)

7.2.2.2.2 Description (7.2.2.2)

04(PCTS_GTI__DEVICE?A:UNTESTED)

A call to *tcdrain*(), on the controlling terminal of the process, returns after all output written to the terminal has been transmitted and returns zero on successful completion.

05(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
A call to *tcdrain*(), on the controlling terminal of the process, from a member of a background process group that is not blocking or ignoring SIGTTOU signals causes a SIGTTOU signal to be sent to its process group. The drain operation is not performed. (See GA60 in 7.2.)
NOTE — When the implementation does not buffer output for the terminal, then the call to tcdrain() only causes a SIGTTOU signal to be sent to its process group.

06(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a member of a background process group that is ignoring SIGTTOU signals calls *tcdrain*() on the controlling terminal of the process, then *tcdrain*() is allowed to perform the actions and returns the value zero, and the SIGTTOU signal is not sent to its process group. (See GA61 in 7.2.)

07(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a member of a background process group that is blocking SIGTTOU signals calls *tcdrain*() on the controlling terminal of the process, then *tcdrain*() is allowed to perform the actions and returns the value zero, and the SIGTTOU signal is not sent to its process group. (See GA62 in 7.2.)

08(PCTS_GTI_DEVICE?A:UNTESTED)

When *fildes* refers to a noncontrolling terminal, then a call to *tcdrain(fildes)* returns after all output written to the terminal has been transmitted and returns zero on successful completion.

Testing Requirements:
If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
Test from a foreground process, a background process that is neither blocking nor ignoring SIGTTOU, a background process that is blocking SIGTTOU, a background process that is ignoring SIGTTOU, and a process that is a member of an orphaned background process group and that is neither blocking nor ignoring SIGTTOU.

7.2.2.2.3 Returns (7.2.2.3)

R01 When a call to *tcdrain*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 4, 6, and 7 in 7.2.2.2.2.)

R02 When a call to *tcdrain*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error (See Assertions 9–11 in 7.2.2.2.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.2.2.2.4 Errors (7.2.2.4)

09(A) When *fildes* is not a valid file descriptor, then a call to *tcdrain(fildes)* returns a value of (*int*)-1 and sets *errno* to [EBADF].

10(A) When the file associated with *fildes* is not a terminal, then a call to *tcdrain(fildes)* returns a value of (*int*)-1 and sets *errno* to [ENOTTY].

11(PCTS_GTI_DEVICE && PCTS_GTI_BUFFERS_OUTPUT?A:UNTESTED)

When a call to *tcdrain(fildes)* is interrupted by a signal, then *tcdrain*() returns a value of (*int*)-1 and sets *errno* to [EINTR].

7.2.2.3 *tcflow* () (7.2.2)

7.2.2.3.1 Synopsis (7.2.2.1)

01(A) If the implementation provides C Standard {2} support:

When the header `<termios.h>` is included, then the function prototype `int tcflow(int, int)` is declared. (See GS36 in 2.7.3.)

Otherwise:

When the header `<termios.h>` is included, then the function *tcflow*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *tcflow*() is defined as a macro when the header `<termios.h>` is included:

When the macro *tcflow*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *tcflow*() is defined as a macro in the header `<termios.h>`:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

7.2.2.3.2 Description (7.2.2.2)

04(A) When the header `<termios.h>` is included, then the symbols TCOOFF, TCOON, TCIOFF, and TCION are defined and are different from each other.

05(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *tcflow(fildes, TCOOFF)*, on the controlling terminal of the process, suspends output, and the return value is zero.

06(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *tcflow(fildes, TCOON)* restarts suspended output, and the return value is zero.

07(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *tcflow(fildes, TCIOFF)* causes the system to transmit a STOP character, and the return value is zero.

Testing Requirements:

Test when data transmission on the line is suspended and not suspended.

08(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *tcflow(fildes, TCION)* causes the system to transmit a START character to restart suspended input, and the return value is zero.

Testing Requirements:

Test when data transmission on the line is suspended and not suspended.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

09(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

A call to *tcflow(fildes, action)*, on the controlling terminal of the process, from a member of a background process group that is not blocking or ignoring SIGTTOU signals causes a SIGTTOU signal to be sent to its process group. The operation specified by action is not performed. (See GA60 in 7.2.)

10(PCTS_GTI__DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} are supported:

When a member of a background process group that is blocking SIGTTOU signals calls *tcflow*() on the controlling terminal of the process, then *tcflow*() is allowed to perform the actions and returns the value zero, and the SIGTTOU signal is not sent to its process group. (See GA61 in 7.2.)

11(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When a member of a background process group that is ignoring SIGTTOU signals calls *tcflow*() on the controlling terminal of the process, then *tcflow*() is allowed to perform the, actions and returns the value zero, and the SIGTTOU signal is not sent to its process group. (See GA62 in 7.2.)

12(PCTS_GTI_DEVICE?A:UNTESTED)

When the first *open*() of a terminal file is done, then neither its input nor its output are suspended.

13(PCTS_GTI_DEVICE?A:UNTESTED)

When *fildes* refers to a noncontrolling terminal, then a call to *tcflow(fildes, TCOOFF)* suspends output, and the return value is zero.

Testing Requirements:

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

Test from a foreground process, a background process that is neither blocking nor ignoring SIGTTOU, a background process that is blocking SIGTTOU, a background process that is ignoring SIGTTOU, and a process that is a member of an orphaned background process group and that is neither blocking nor ignoring SIGTTOU.

7.2.2.3.3 Returns (7.2.2.3)

R01 When a call to *tcflow*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 5-8, 10, and 11 in 7.2.2.3.2.)

R02 When a call to *tcflow*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error (See Assertions 14–16 in 7.2.2.3.4.)

7.2.2.3.4 Errors (7.2.2.4)

14(A) When *fildes* is not a valid file descriptor, then a call to *tcflow(fildes, action)* returns a value of (*int*)-1 and sets *errno* to [EBADF].

15(PCTS_GTI_DEVICE?A:UNTESTED)

When *action* does not have a proper value, then a call to *tcflow(fildes, action)* returns a value of (*int*)-1 and sets *errno* to [EINVAL].

16(A) When the file associated with *fildes* is not a terminal, then a call to *tcflow(fildes, action)* returns a value of (*int*)-1 and sets *errno* to [ENOTTY].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

7.2.2.4 *tcflush* () (7.2.2)

7.2.2.4.1 Synopsis (7.2.2.1)

01(A) If the implementation provides C Standard {2} support:

When the header <termios.h> is included, then the function prototype `int tcflush(int, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <termios.h> is included, then the function *tcflush*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *tcflush*() is defined as a macro when the header <termios.h> is included:

When the macro *tcflush*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *tcflush*() is defined as a macro in the header <termios.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

7.2.2.4.2 Description (7.2.2.2)

04(A) When the header <termios.h> is included, then the symbols TCIFLUSH, TCOFLUSH, and TCIOFLUSH are defined and are different from each other.

05(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *tcflush(fildes, TCIFLUSH)*, on the controlling terminal of the process, discards data received from the terminal but not read and returns the value zero.

06(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *tcflush(fildes, TCOFLUSH)* discards data written to the terminal but not transmitted and returns the value zero.

NOTE — When the implementation does not buffer output for the terminal, then the call to *tcflush*() only returns the value zero.

07(PCTS_GTI_DEVICE?A:UNTESTED)

A call to *tcflush(fildes, TCIOFLUSH)* discards both data received from the terminal but not read and data written to the terminal but not transmitted, and returns the value zero.

NOTE — When the implementation does not buffer output for the terminal, then the call to *tcflush*() only discards data received from the terminal but not read and returns the value zero.

08(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

A call to *tcflush*(), on the controlling terminal of the process, from a member of a background process group that is not blocking or ignoring SIGTTOU signals causes a SIGTTOU signal to be sent to its process group. The flush operation is not performed. (See GA60 in 7.2.)

NOTE — When the implementation does not buffer output for the terminal, then the call to *tcflush*() only causes a SIGTTOU signal to be sent to its process group and does not discard data received from the terminal but not read.

09(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When a member of a background process group that is blocking SIGTTOU signals calls *tcflush*() on the controlling terminal of the process, then *tcflush*() is allowed to perform the actions and returns the value zero, and the SIGTTOU signal is not sent to its process group. (See GA61 in 7.2.)

NOTE — When the implementation does not buffer output for the terminal, then the call to *tcflush*() is allowed only to discard data received from the terminal but not read and to return the value zero.

10(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When a member of a background process group that is ignoring SIGTTOU signals calls *tcflush*() on the controlling terminal of the process, then *tcflush*() is allowed to perform the actions and returns the value zero, and the SIGTTOU signal is not sent to its process group. (See GA62 in 7.2.)

NOTE — When the implementation does not buffer output for the terminal, then the call to *tcflush*() is allowed only to discard data received from the terminal but not read and to return the value zero.

11(PCTS_GTI_DEVICE?A:UNTESTED)

When *fildes* refers to a noncontrolling terminal, then a call to *tcflush(fildes, TCIFLUSH)* discards data received from the terminal but not read and returns the value zero.

Testing Requirements:

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

Test from a foreground process, a background process that is neither blocking nor ignoring SIGTTOU, a background process that is blocking SIGTTOU, a background process that is ignoring SIGTTOU, and a process that is a member of an orphaned background process group and that is neither blocking nor ignoring SIGTTOU.

7.2.2.4.3 Returns (7.2.2.3)

R01 When a call to *tcflush*() completes successfully, then a value of (*int*)0 is returned. (See Assertions 5-7, 9, and 10 in 7.2.2.4.2.)

R02 When a call to *tcflush*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error (See Assertions 12–14 in 7.2.2.4.4.)

7.2.2.4.4 Errors (7.2.2.4)

12(A) When *fildes* is not a valid file descriptor, then a call to *tcflush(fildes, queue_selector)* returns a value of (*int*)-1 and sets *errno* to [EBADF].

13(PCTS_GTI_DEVICE?A:UNTESTED)

When *queue_selector* does not have a proper value, then a call to *tcflush(fildes, queue_selector)* returns a value of (*int*)-1 and sets *errno* to [EINVAL].

14(A) When the file associated with *fildes* is not a terminal, then a call to *tcflush(fildes, queue_selector)* returns a value of (*int*)-1 and sets *errno* to [ENOTTY].

7.2.3 *tcgetpgrp*()

7.2.3.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <unistd.h> is included, then the function prototype `pid_t tcgetpgrp(int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <unistd.h> is included, then the function *tcgetpgrp*() is declared with the result type *pid_t*. (See GA36 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

02(C) If *tcgetpgrp*() is defined as a macro when the header <unistd.h> is included:
When the macro *tcgetpgrp*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *pid_t*. (See GA37 in 2.7.3.)

03(C) If *tcgetpgrp*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

7.2.3.2 Description

D01(C) If the behavior associated with {_POSIX_JOB_CONTROL} being set is not supported, and if it is documented whether *tcgetpgrp*() is supported:
The details are contained in 7.2.3.2 of the PCD.1.

04(PCTS_GTI_D EVICE?C:UNTESTED)

If either the behavior associated with {_POSIX_JOB_CONTROL} or *tcgetpgrp*() is supported:
A call to *tcgetpgrp*() returns the value of the process group ID of the foreground process group associated with the terminal.
Testing Requirements:
If the behavior associated with {_POSIX_JOB_CONTROL} is supported, test behavior of a process that is a member of a foreground process group and a process that is a member of a background process group.

7.2.3.3 Returns

R01 When a call to *tcgetpgrp*() completes successfully, then the process group ID of the foreground process group associated with the terminal is returned. (See Assertion 4 in 7.2.3.2.)

05(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When a call to *tcgetpgrp*() is made from a process that is a member of a session that has no foreground process group, then *tcgetpgrp*() returns a value greater than 1 that does not match the process group ID of any existing process.

R02 When a call to *tcgetpgrp*() completes unsuccessfully, then a value of (*pid_t*)-1 is returned and sets *errno* to indicate the error (See Assertions 6-9 in 7.2.3.4.)

7.2.3.4 Errors

06(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *tcgetpgrp*() is supported:
When *fildes* is not a valid file descriptor, then a call to *tcgetpgrp(fildes)* returns a value of (*pid_t*)-1 and sets *errno* to [EBADF].

07(C) If both the behavior associated with {_POSIX_JOB_CONTROL} and *tcgetpgrp*() are not supported:
A call to *tcgetpgrp*() is unsuccessful, returns a value of (*pid_t*)-1, and sets *errno* to [ENOSYS].

08(PCTS_GTI_DEVICE?C:UNTESTED)

If either the behavior associated with {_POSIX_JOB_CONTROL} or *tcgetpgrp*() is supported:
When the calling process does not have a controlling terminal, then a call to *tcgetpgrp(fildes)* returns a value of (*pid_t*)-1 and sets *errno* to [ENOTTY].

09(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *tcgetpgrp*() is supported:
When the file associated with *fildes* is not the controlling terminal of the calling process, then a call to *tcgetpgrp(fildes)* returns a value of (*pid_t*)-1 and sets *errno* to [ENOTTY].
Testing Requirements:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860364 918 ■

Test for a file that is not a terminal and, if the implementation provides device types that support the general terminal interface, for a terminal that is not the controlling terminal of the process.

7.2.4 *tcsetpgrp*()

7.2.4.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <unistd.h> is included, then the function prototype `int tcsetpgrp(int, pid_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <unistd.h> is included, then the function *tcsetpgrp*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *tcsetpgrp*() is defined as a macro when the header <unistd.h> is included:
When the macro *trsetpgrp*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *tcsetpgrp*() is defined as a macro in the header <unistd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

7.2.4.2 Description

D01(C) If the behavior associated with {_POSIX_JOB_CONTROL} being set is not supported, and if it is documented whether *tcsetpgrp*() is supported:
The details are contained in 7.2.4.2 of the PCD.1.

04(PCTS_GTI_DEVICE?C:UNTESTED)

If either the behavior associated with {_POSIX_JOB_CONTROL} or *tcsetpgrp*() is supported:
When the file associated with *fildes* is the controlling terminal of the calling process, when the controlling terminal is associated with the session of the calling process, and when *pgrp_id* is the process group ID of a process in the same session as the calling process, then a call to *tcsetpgrp(fildes, pgrp_id)* sets the foreground process group ID associated with the terminal to *pgrp_id* and returns a value of zero.

7.2.4.3 Returns

R01 When a call to *tcsetpgrp*() completes successfully, then a value of (*int*)0 is returned. (See Assertion 4 in 7.2.4.2.)

R02 When a call to *tcsetpgrp*() completes unsuccessfully, then a value of (*int*)-1 is returned and sets *errno* to indicate the error. (See Assertions 5–11 in 7.2.4.4.)

7.2.4.4 Errors

05(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *tcsetpgrp*() is supported:
When *fildes* is not a valid file descriptor, then a call to *tcsetpgrp(fildes, pgrp_id)* returns a value of (*int*)-1 and sets *errno* to [EBADF].

06(PCTS_GTI_DEVICE?C:UNTESTED)

If either the behavior associated with {_POSIX_JOB_CONTROL} or *tcsetpgrp*() is supported, and the implementation supports an invalid value for *pgrp_id*:

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When *pgrp_id* is a value not supported by the implementation, then a call to *tcsetpgrp(fildes, pgrp_id)* returns a value of (*int*)-1, sets *errno* to [EINVAL], and does not change the process group ID of the controlling terminal associated with *fildes*.

07(C) If both the behavior associated with {_POSIX_JOB_CONTROL} and *tcsetpgrp*() are not supported:
A call to *tcsetpgrp*() is unsuccessful, returns a value of (*int*)-1, and sets *errno* to [ENOSYS].

08(PCTS_GTI_DEVICE?C:UNTESTED)

If either the behavior associated with {_POSIX_JOB_CONTROL} or *tcsetpgrp*() is supported:
When the calling process does not have a controlling terminal, then a call to *tcsetpgrp(fildes, pgrp_id)* returns a value of (*int*)-1, sets *errno* to [ENOTTY], and does not change the process group ID of the controlling terminal associated with *fildes*.

09(C) If either the behavior associated with {_POSIX_JOB_CONTROL} or *tcsetpgrp*() is supported:
When *fildes* is not associated with the controlling terminal, then a call to *tcsetpgrp(fildes, pgrp_id)* returns a value of (*int*)-1 and sets *errno* to [ENOTTY].
Testing Requirements:
Test for a file that is not a terminal and, if the implementation provides device types that support the general terminal interface, for a terminal that is not the controlling terminal of the process.

10(PCTS_GTI_DEVICE?C:UNTESTED)

If either the behavior associated with {_POSIX_JOB_CONTROL} or *tcsetpgrp*() is supported:
When the controlling terminal is no longer associated with the session of the calling process, then a call to *tcsetpgrp(fildes, pgrp_id)* returns a value of (*int*)-1, sets *errno* to [ENOTTY], and does not change the process group ID of the controlling terminal associated with *fildes*.

11(PCTS_GTI_DEVICE?C:UNTESTED)

If either the behavior associated with {_POSIX_JOB_CONTROL} or *tcsetpgrp*() is supported:
When the value of *pgrp_id* does not match the process group ID of a process in the same session as the calling process, then a call to *tcsetpgrp(fildes, pgrp_id)* returns a value of (*int*)-1, sets *errno* to [EPERM], and does not change the process group ID of the controlling terminal associated with *fildes*.

8. Language-Specific Services for the C Programming Language

8.1 Referenced C Language Routines

NOTE — Implementations claiming conformance to Common-Usage C shall adhere to the requirements stated in 1.4.19.

D01(C) If the implementation provides Common-Usage C support and does not support a C Standard {2} function or deviates from the C Standard {2}:
These differences are documented in the appropriate section of the PCD.1.

GA63 Implementations shall comply with the requirements outlined for the C Standard {2}.

8.1.1 *assert*() (8.1)

8.1.1.1 Synopsis (8.1)

01(A) When the header <assert.h> is included, then *assert*() is defined as a macro that expands to an expression with the result type *void*. (See GA36 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.1.2 Description (8.1)

02(B) If the implementation provides C Standard {2} support:
Element *assert*() complies with the requirements of the C Standard {2}. (See GA63 in 8.)

Otherwise:
Element *assert*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8..)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.1.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.1.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.2 Character Handling Functions (8.1)

8.1.2.1 Synopsis (8.1)

01(A) For *funct*() of *isalnum*(), *isalpha*(), *iscntrl*(), *isdigit*(), *isgraph*(), *islower*(), *isprint*(), *ispunct*(), *isspace*(), *isupper*(), *isxdigit*(), *tolower*(), and *toupper*():
If the implementation provides C Standard {2} support:
When the header <ctype.h> is included, then the function prototype `int funct(int)` is declared. (See GA36 in 2.7.3.)
Otherwise:
When the header <ctype.h> is included, then the function *funct*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) For *funct*() of *isalnum*(), *isalpha*(), *iscntrl*(), *isdigit*(), *isgraph*(), *islower*(), *isprint*(), *ispunct*(), *isspace*(), *isupper*(), *isxdigit*(), *tolower*(), and *toupper*():
If *funct*() is defined as a macro when the header <ctype.h> is included:
When the macro *funct*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If the *isalnum*(), *isalpha*(), *isdigit*(), *isgraph*(), *islower*(), *isprint*(), *ispunct*(), *isspace*(), *isupper*(), *isxdigit*(), *tolower*(), or *toupper*() function is defined as a macro in the header <ctype.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and yields a result value that can be used in complex expressions without requiring extra parentheses. (See GA01 in 1.3.4.)

8.1.2.2 Description (8.1)

04(B) For *funct*() of *isalnum*(), *isalpha*(), *iscntrl*(), *isdigit*(), *isgraph*(), *islower*(), *isprint*(), *ispunct*(), *isspace*(), *isupper*(), *isxdigit*(), *tolower*(), and *toupper*():
If the implementation provides C Standard {2} support:
Element *funct*() complies with the requirements of the C Standard {2}. (See GA63 in 8..)
Otherwise:
Element *funct*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8.)
See Reason 2 in Section 5. of POSIX.3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.2.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.2.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.3 *setlocale*() (8.1)

8.1.3.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <locale.h> is included, then the function prototype
`char * setlocale(int, const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <locale.h> is included, then the function *setlocale*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

02(C) If *setlocale*() is defined as a macro when the header <locale.h> is included:
When the macro *setlocale*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

03(C) If *setlocale*() is defined as a macro in the header <locale.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.3.2 Description (8.1.2.2)

D01(C) If the implementation supports values for *locale* other than "C":
The other values supported for *locale* are documented in 8.1.2.2 of the PCD.1.

04(A) When *locale* is a pointer to the string "C", then a call to *setlocale*(*category, locale*) sets the locale for the specified category to the minimal environment for C-language translation.

NOTE — The minimal environment for C-language translation is specified in Section 4.4 of the C Standard {2}.

05(A) When *locale* is a pointer to the string "POSIX", then a call to *setlocale*(*category, locale*) sets the locale for the specified category to an environment that includes the minimal environment, for C-language translation.

06(A) When no call has been made to *setlocale*() by the application after the most recent successful replacement of the process image, then the "C" locale is the locale associated with the process.

D02(C) If the string returned from *setlocale*(), when *locale* is a **NULL** pointer, is documented:
The details are contained in 8.1.2.2 of the PCD.1.

07(A) When *locale* is a pointer to a NULL string, and when the environment variable **LC_ALL** specifies a valid locale, then *setlocale*(*category, locale*) sets the locale for the requested category (or all categories in the case of category **LC_ALL**) to the value of the environment variable **LC_ALL**.

08(A) When *locale* is a pointer to a NULL string, and when the environment variable **LC_ALL** is not a NULL string and does not specify a valid locale, then *setlocale*(*category, locale*) returns NULL and does not change the locale setting.

D03(C) If additional categories are supported for the *setlocale*() function:
These categories are documented in 8.1.2.2 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D04(A) When the **LC_ALL** environment variable is not set or is set to the empty string, when the environment variable corresponding to the specified category is not set or is set to the null string, and when the **LANG** environment variable is not set or is set to the empty string, whether *setlocale*() sets the specified category of the locale of the process to a system wide default value, to "C", or to "POSIX" is stated in 8.1.2.2 of the PCD.1.

09(A) When *category* is any of **LC_CTYPE**, **LC_COLLATE**, **LC_TIME**, **LC_NUMERIC**, or **LC_MONETARY**, when *locale* is a pointer to a NULL string, when the environment variable **LC_ALL** is either not set or is set to a NULL string, and when the environment variable corresponding to *category* is set to a valid locale, then *setlocale*(*category, locale*) sets the specified category to the value in the corresponding environment variable.

10(A) When *category* is any of **LC_CTYPE**, **LC_COLLATE**, **LC_TIME**, **LC_NUMERIC**, or **LC_MONETARY**, when *locale* is a pointer to a NULL string, when the environment variable **LC_ALL** is either not set or is set to a NULL string, and when the environment variable corresponding to *category* is not a NULL string and is not a valid locale, then *setlocale*(*category, locale*) returns NULL and does not change the locale setting.

11(A) When *category* is any of **LC_CTYPE**, **LC_COLLATE**, **LC_TIME**, **LC_NUMERIC**, or **LC_MONETARY**, when *locale* is a pointer to a NULL string, when the environment variable **LC_ALL** and the environment variable corresponding to *category* are each either not set or set to a NULL string, and when the environment variable **LANG** is set to a valid locale, then *setlocale*(*category, locale*) sets the specified category to the value in the **LANG** environment variable.

12(A) When *category* is any of **LC_CTYPE**, **LC_COLLATE**, **LC_TIME**, **LC_NUMERIC**, or **LC_MONETARY**, when *locale* is a pointer to a NULL string, when the environment variable **LC_ALL** is either not set or is set to NULL, when the corresponding environment variable is either not set or is set to a NULL string, and when the environment variable **LANG** is not a NULL string and does not specify a valid locale, then *setlocale*(*category, locale*) returns NULL and does not change the locale setting.

13(A) When *category* is **LC_ALL**, when locale is a pointer to a NULL string, and when the environment variable **LC_ALL**. is either not set or is set to a NULL string, then for each of the environment variables **LC_CTYPE**, **LC_COLLATE**, **LC_TIME**, **LC_NUMERIC**, and **LC_MONETARY** that are not NULL strings and that specify a valid locale, *setlocale*(*category, locale*) sets the locale for that *category* to the value of the corresponding environment variable.

14(A) When *category* is **LC_ALL**, when *locale* is a pointer to a NULL string, when the environment variable **LC_ALL** is either not set or is set to a NULL string, and when the environment variable **LANG** is set to a valid locale, then, for each of the environment variables **LC_CTYPE**, **LC_COLLATE**, **LC_TIME**, **LC_NUMERIC**, and **LC_MONETARY** that is either not set or set to a NULL string, *setlocale*(*category, locale*) sets the *locale* for that category to the value of the environment variable **LANG**.

15(A) When *category* is **LC_ALL**, when *locale* is a pointer to a NULL string, when the environment variable **LC_ALL** is either not set or is set to NULL, and when any of the environment variables **LC_CTYPE**, **LC_COLLATE**, **LC_TIME**, **LC_NUMERIC**, and **LC_MONETARY** is not a NULL string and does not specify a valid locale, then *setlocale*(*category, locale*) returns NULL and does not change the locale setting.

16(A) When *category* is **LC_ALL**, when *locale* is a pointer to a NULL string, when the environment variable **LC_ALL** is either not set or is set to a NULL, when any of the environment variables **LC_CTYPE**, **LC_COLLATE**, **LC_TIME**, **LC_NUMERIC**, and **LC_MONETARY** is either not set or is set to a NULL string, and when the environment variable **LANG** is not a NULL string and does not specify a valid locale, then *setlocale*(*category, locale*) returns NULL and does not change the locale setting.

8.1.3.3 Returns (8.1)

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.3.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.4 Mathematics Functions (8.1)

8.1.4.1 Synopsis (8.1)

01(A) For *funct*() of *acos*(), *asin*(), *atan*(), *cos*(), *sin*(), *tan*(), *cosh*(), *sinh*(), *tanh*(), *exp*(), *log*(), *log10*(), *sqrt*(), *ceil*(), *fabs*(), and *floor*():

If the implementation provides C Standard {2} support:

When the header <math.h> is included, then the function prototype `double funct(double)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <math.h> is included, then the function *funct*() is declared with the result type *double*. (See GA36 in 2.7.3.)

For *funct*() of *atan2*(), *pow*(), and *fmod*():

If the implementation provides C Standard {2} support:

When the header <math.h> is included, then the function prototype `double funct(double, double)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <math.h> is included, then the function *funct*() is declared with the result type *double*. (See GA36 in 2.7.3.)

For *frexp*():

If the implementation provides C Standard {2} support:

When the header <math.h> is included, then the function prototype `double frexp(double, int *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <math.h> is included, then the function *frexp*() is declared with the result type *double*. (See GA36 in 2.7.3.)

For *ldexp*():

If the implementation provides C Standard {2} support:

When the header <math.h> is included, then the function prototype `double ldexp(double, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <math.h> is included, then the function *ldexp*() is declared with the result type *double*. (See GA36 in 2.7.3.)

For *modf*():

If the implementation provides C Standard {2} support:

When the header <math.h> is included, then the function prototype `double modf(double, double *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <math.h> is included, then the function *modf*() is declared with the result type *double*. (See GA36 in 2.7.3.)

02(C) For *funct*() of *acos*(), *asin*(), *atan*(), *cos*(), *sin*(), *tan*(), *cosh*(), *sinh*(), *tanh*(), *exp*(), *log*(), *log10*(), *sqrt*(), *ceil*(), *fabs*(), *floor*(), *atan2*(), *pow*(), *fmod*(), *frexp*(), *ldexp*(), and *modf*():

If *funct*() is defined as a macro when the header <math.h> is included:

When the macro *funct*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type double. (See GA37 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

03(C) If the *acos*(), *asin*(), *atan*(), *cos*(), *sin*(), *tan*(), *cosh*(), *sinh*(), *tanh*(), *exp*(), *log*(), *log10*(), *sqrt*(), *ceil*(), *fabs*(), *floor*(), *atan2*(), *pow*(), *fmod*(), *frexp*(), *ldexp*(), or *modf*() function is defined as a macro in the header <math.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary and yields a result value that can be used in complex expressions without requiring extra parentheses. (See GA01 in 1.3.4.)

8.1.4.2 Description (8.1)

04(B) For *funct*() of *acos*(), *asin*(), *atan*(), *cos*(), *sin*(), *tan*(), *cosh*(), *sinh*(), *tanh*(), *exp*(), *log*(), *log10*(), *sqrt*(), *ceil*(), *fabs*(), *floor*(), *atan2*(), *pow*(), *fmod*(), *frexp*(), *ldexp*(), and *modf*():

If the implementation provides C Standard {2} support:

Element *funct*() complies with the requirements of the C Standard {2}. (See GA63 in 8.)

Otherwise:

Element *funct*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8..)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.4.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.4.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.5 *setjmp*() (8.1)

8.1.5.1 Synopsis (8.1)

01(C) If *setjmp*() is not defined as a macro:

When the header <setjmp.h> is included, then *setjmp*() is declared as an identifier with external linkage and result type *int*. (See GA36 in 2.7.3.)

02(C) If *setjmp*() is defined as a macro in the header <setjmp.h>:

When the header <setjmp.h> is included and the macro *setjmp*() is invoked with the correct argument types, then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *setjmp*() is defined as a macro in the header <setjmp.h>:

It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.5.2 Description (8.1)

04(B) If the implementation provides C Standard {2} support:

Element *setjmp*() complies with the requirements of the C Standard {2}. (See GA63 in 8..)

Otherwise:

Element *setjmp*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8.)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.5.3 Returns (8.1)

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.5.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.6 *longjmp*() (8.1)

8.1.6.1 Synopsis (8.1)

01(C) If the implementation provides C Standard {2} support:
When the header <setjmp.h> is included, then the function prototype
void longjmp(jmp_buf, int) is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *longjmp*() is contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(D) If the implementation provides C Standard {2} support and *longjmp*() is defined as a macro when the header <setjmp.h> is included:
When the macro *longjmp*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)

See Reason 2 in Section 5. of POSIX.3 {4}.

03(C) If *longjmp*() is defined as a macro in the header <setjmp.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 in 1.3.3.)

8.1.6.2 Description (8.1)

04(B) If the implementation provides C Standard {2} support:
Element *longjmp*() complies with the requirements of the C Standard {2}. (See GA63 in 8.)

Otherwise:
Element *longjmp*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8..)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.6.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.6.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.7 *clearerr*() (8.1)

8.1.7.1 Synopsis (8.1)

01(C) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype
void clearerr(FILE *) is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *clearerr*() is contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

02(D) If the implementation provides C Standard {2} support and *clearerr*() is defined as a macro when the header <stdio.h> is included:

When the macro *clearerr*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)

See Reason 2 in Section 5. of POSIX.3 {4}.

03(C) If *clearerr*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 in 1.3.3.)

8.1.7.2 Description (8.1)

04(B) If the implementation provides C Standard {2} support:

Element *clearerr*() complies with the requirements of the C Standard {2}. (See GA63 in 8..)

Otherwise:

Element *clearerr*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8..)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.7.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.7.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.8 *fclose*() (8.1)

8.1.8.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype `int fclose(FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *fclose*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *fclose*() is defined as a macro when the header <stdio.h> is included:

When the macro *fclose*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *fclose*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.8.2 Description (8.2.3.2)

04(A) The *fclose*() function deallocates the file descriptor underlying the stream indicated by its argument (i.e., this file descriptor will be made available for subsequent *open*()s by the process), and a value of zero is returned.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

05(A) When a call to *fclose*() on the stream is performed, then all outstanding record locks owned by the process on the file associated with the stream are removed.

R01 When a call to *fclose*() is interrupted by a signal that is to be caught, then the call to *fclose*() returns a value of (*int*)−1 and sets *errno* to [EINTR]. (See Assertion 15 in 8.1.8.4.)

06(B) When all file descriptors associated with a pipe have been closed by *fclose*(), then data remaining in the pipe is discarded.

See Reason 3 in Section 5. of POSIX.3 [4].

07(A) When all file descriptors associated with a FIFO special file have been closed by *fclose*(), then data remaining in the FIFO is discarded.

08(B) When all file descriptors associated with an open file description have been closed by *fclose*(), then the open file description is freed.

See Reason 1 in Section 5. of POSIX.3 [4].

09(B) When all file descriptors associated with a file have been closed by *fclose*(), and when the link count of the file is zero, then the space occupied by the file is freed.

See Reason 1 in Section 5. of POSIX.3 [4].

10(B) When all file descriptors associated with a file have been closed by *fclose*(), and when the link count of the file is zero, then the file is no longer accessible.

See Reason 3 in Section 5. of POSIX.3 [4].

11(A) When a file is closed via *fclose*() by the last process that had it open, all time-related fields still marked for update are updated.

12(A) When the stream is writable and buffered data is caused to be written, then a call to *fclose*(*stream*) marks for update the *st_ctime* and *st_mtime* fields of the underlying file.

8.1.8.3 Returns (8.2.3.11)

R02 When a call to *fclose*() completes unsuccessfully, then a value of EOF is returned and sets *errno* to indicate the error. (See Assertions 13-19 in 8.1.8.4.)

8.1.8.4 Errors (8.2.3.11)

3(C) If the implementation provides buffered streams:
When the stream references a pipe or FIFO, and when the underlying file descriptor has the O_NONBLOCK flag set and contains insufficient capacity to accept the buffered data, then a call to *fclose*() returns a value of EOF and sets *errno* to [EAGAIN].
Testing Requirements:
Test for both a pipe and a FIFO.

14(A) When the stream pointer argument addresses an underlying file descriptor that is closed, then a call to *fclose*() returns a value of EOF and sets *errno* to [EBADF].

15(C) If the implementation provides buffered streams:
When *fclose*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *fclose*() returns a value of EOF and sets *errno* to [EINTR].

16(D) If the implementation provides buffered streams:
When attempting to write to a file, and when flushing the output buffer would cause the size of the file to exceed an implementation-defined maximum file size, then a call to *fclose*() returns a value of EOF and sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX.3 [4].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

17(D) If the implementation provides buffered streams:
When attempting to write to a device, and when flushing the output buffer would exceed the amount of space available for data on that device, then a call to *fclose*() returns a value of EOF and sets *errno* to [ENOSPC].

See Reason 1 in Section 5. of POSIX.3 {4}.

18(C) If the implementation provides buffered streams:
When attempting to write to a pipe or a FIFO that is not open for reading by any process, then a call to *fclose*() returns a value of EOF, sets *errno* to [EPIPE], and sends a SIGPIPE signal to the calling process.
Testing Requirements:
Test for both a pipe and a FIFO.

19(PCTS_GTI_DEVICE?C:UNTESTED)

If both the behavior associated with {_POSIX_JOB CONTROL} is supported and the implementation provides buffered streams:
When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, and when TOSTOP is set, then a call to *fclose*() returns a value of EOF and sets *errno* to [EIO].

8.1.9 *feof*() (8.1)

8.1.9.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `int feof(FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *feof*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *feof*() is defined as a macro when the header <stdio.h> is included:
When the macro *feof*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *feof*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.9.2 Description (8.1)

04(A) A call to *feof*(*stream*) returns nonzero when the end of file indicator associated with stream is set.

05(A) A call to *feof*(*stream*) returns zero when the end of file indicator associated with stream is clear.

8.1.9.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.9.4 Errors (8.1)

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.10 *ferror*() (8.1)

8.1.10.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype `int ferror(FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *ferror*() is either declared with result type int or not declared in the header. (see GA36 in 2.7.3.)

02(C) If *ferror*() is defined as a macro when the header <stdio.h> is included:

When the macro *ferror*() is invoked with the correct, argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (see GA37 in 2.7.3.)

03(C) If *ferror*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.10.2 Description (8.1)

04(A) A call to *ferror*(*stream*) returns nonzero when the error indicator associated with *stream* is set.

05(A) A call to *ferror*(*stream*) returns zero when the error indicator associated with *stream* is clear.

8.1.10.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.10.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.11 *fflush*() (8.1)

8.1.11.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype `int fflush(FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *fflush*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *fflush*() is defined as a macro when the header <stdio.h> is included:

When the macro *fflush*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *fflush*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protracted by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.11.2 Description (8.2.3.4)

04(A) When the stream is writable and buffered data is caused to be written, then a call to *fflush*() marks for update the *st_ctime* and *st_mtime* fields of the underlying file.

8.1.11.3 Returns (8.2.3.11)

R01 When a call to *fflush*() completes unsuccessfully, then a value of EOF is returned and sets *errno* to indicate the error. (See Assertions 5-11 in 8.1.11.4.)

8.1.11.4 Errors (8.2.3.11)

05(C) If the implementation provides buffered streams:

When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient capacity to accept the buffered data, then a call to *fflush*() returns a value of EOF and sets *errno* to [EAGAIN].

Testing Requirements:

Test for both a pipe and a FIFO.

06(A) When the stream pointer argument addresses a file descriptor that is closed, then a call to *fflush*() returns a value of EOF and sets *errno* to [EBADF].

07(C) If the implementation provides buffered streams:

When *fflush*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *fflush*() returns a value of EOF and sets *errno* to [EINTR].

08(D) If the implementation provides buffered streams:

When attempting to write to a file that exceeds an implementation-defined maximum file size, then a call to *fflush*() returns a value of EOF and sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX.3 {4}.

09(D) If the implementation provides buffered streams:

When attempting to write to a device that has no more space for en a call to *fflush*() returns a value of EOF and sets *errno* to [ENOSPC].

See Reason 1 in Section 5. of POSIX.3 {4}.

10(C) If the implementation provides buffered streams:

When attempting to write to a pipe or a FIFO t hat is not open for reading by any process, then a call to *fflush*() returns a value of EOF, sets *errno* to [EPIPE], and sends a SIGPIPE signal to the calling process.

Testing Requirements:

Test for both a pipe and a FIFO.

11(PCTS_GTI_DEVICE?C:UNTESTED)

If both the behavior associated with {_POSIX_JOB_CONTROL} is supported and the implementation provides buffered streams:

When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, and when TOSTOP is set, then a call to *fflush*() returns a v alue of EOF and sets *errno* to [EIO].

8.1.12 *fgetc*() (8.1)

8.1.12.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the header <stdio.h> is included, then the function prototype `int fgetc(FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *fgetc*() is either declared with result type *int* not declared in the header. (See GA36 in 2.7.3.)

02(C) If *fgetc*() is defined as a macro when the header <stdio.h> is included:

When the macro *fgetc*() is invoked with the correct argument types (or compatible argument types in the case that C Standard 2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *fgetc*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (see GA01 in 1.3.4.)

8.1.12.2 Description (8.2.3.5)

04(A) When data was not supplied by a prior call to *ungetc*(), and when there has been no previous buffered read operation on the stream, then a call to *fgetc*() marks for update the *st_atime* field of the underlying file.

8.1.12.3 Returns (8.2.3.11)

R01 When a call to *fgetc*() completes unsuccessfully, then a value of EOF is returned and sets *errno* to indicate the error. (See Assertions 5-9 in 8.1.12.4.)

8.1.12.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient data to process a read request, then a call to *fgetc*() returns a value of EOF and sets *errno* to [EAGAIN].

Testing Requirements:

Test for both a pipe and a FIFO.

06(A) When the stream pointer argument addresses a file descriptor that is not open for reading, then a call to *fgetc*() returns a value of EOF and sets *errno* to [EBADF].

07(A) When *fgetc*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *fgetc*() returns a value of EOF and sets *errno* to [EINTR].

08(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process is ignoring or blocking the SIGTTIN signal and is a member of a background process group, and when the *stream* argument references the controlling terminal, then a call to *fgetc*(*stream*) returns a value of EO and sets *errno* to [EIO].

09(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process is a member of an orphaned background process group, and when the *stream*argument references the controlling terminal, then a call to *fgetc*(*stream*) returns a value of EOF and sets *errno* to [EIO].

8.1.13 *fgets*() (8.1)

8.1.13.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the header <stdio.h> is included, then the function prototype
`char * fgets(char *,int, FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *fgets*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

02(C) If *fgets*() is defined as a macro when the header <stdio.h> is included:

When the macro *fgets*() is invoked with the correct argument types (or compatible argument types in the case that C Standard 2} support is provided), then it expands to an expression with the result type *char**. (See GA37 in 2.7.3.)

03(C) If *fgets*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.13.2 Description (8.2.3.5)

04(A) On a call to *fgets*() when data was not supplied by a prior call to *ungetc*(), and when there has been no previous buffered read operation on the stream, then the *st_atime* field of the underlying file is marked for update.

8.1.13.3 Returns (8.2.3.11)

R01 When a call to *fgets*() completes unsuccessfully, then a **NULL** pointer is returned and sets *errno* to indicate the error. (See Assertions 5-9 in 8.1.13.4.)

8.1.13.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient data to process a read request, then a call to *fgets*() returns a **NULL** pointer and sets *errno* to [EAGAIN].

Testing Requirements:

Test for both a pipe and a FIFO.

06(A) When the stream pointer argument addresses a file descriptor that is not open for reading, then a call to *fgets*() returns a **NULL** pointer and sets *errno* to [EBADF].

07(A) When *fgets*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *fgets*() returns a **NULL** pointer and sets *errno* to [EINTR].

08(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process is ignoring or blocking the SIGTTIN signal and is a member of a background process group, and when the *stream* argument references the controlling terminal, then a call to *fgets*(*stream*) returns a **NULL** pointer and sets *errno* to [EIO].

09(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process is a member of an orphaned background process group, and when the *stream* argument references the controlling terminal, then a call to *fgets*(*stream*) returns a **NULL** pointer and sets *errno* to [EIO].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.14 *fopen*() (8.1)

8.1.14.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype
`FILE * fopen(const char *, const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *fopen*() is declared with the result type *FILE* *. (See GA36 in 2.7.3.)

02(C) If *fopen*() is defined as a macro when the header <stdio.h> is included:

When the macro *fopen*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *FILE* *. (See GA37 in 2.7.3.)

03(C) If *fopen*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.14.2 Description (8.2.3.1)

04(A) When the first filename component of the *filename* argument is ".", and the pathname does not begin with a slash, then *fopen*() resolves the pathname by locating the second filename component when specified) in the current working directory. (See GA14 in 2.3.6.)

05(A) When the *filename* argument points to the string "/", then *fopen*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

06(A) When the *filename* argument points to the string "///", then *fopen*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

07(A) When the *filename* argument points to a string beginning with a single slash or beginning with three or more slashes, then *fopen*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *filename* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *fopen*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *filename* argument points to the string "F1/" and `F1` is a directory, then *fopen*() resolves the pathname to `F1`. (See GA19 in 2.3.6.)

10(A) When the argument *filename* points to the string "F1//" and `F1` is a directory, then *fopen*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *filename* argument points to the string "F1/F2", then *fopen*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the pathname argument points to the string "F1/./F2", then *fopen*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *filename* argument points to the string "F1/../F1/F2", then *fopen*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *filename* argument points to the string "F1//F2", then *fopen*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:
When the *filename* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *fopen*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *fopen*(*filename*, *type*), the pathname *filename* supports *filenames* containing any of the following characters:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghij klmnopqrstuvwxyz
0123456789._-

(See GA02 in 2.2.2.32.)

17(A) On a call to *fopen*(*filename*, *type*), the pathname *filename* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) A call to *fopen*() allocates a file descriptor as a call to *open*() does.

19(A) When a call to *fopen*() creates a file, then the file permission bits are set to allow both reading and writing for owner, for group and for other users except for those bits set in the file mode creation mask of the process. No execute (search) permission bits are set.

8.1.14.3 Returns (8.2.3.11)

R01 When a call to *fopen*() completes unsuccessfully, then a **NULL** pointer is returned and sets *errno* to indicate the error. (See Assertions 20-24, 26, and 29-34 in 8.1.14.4.)

8.1.14.4 Errors (8.2.3.11)

20(A) When search permission is denied on a component of the *filename* prefix, then a call to *fopen*(*filename*, *type*) returns a **NULL** pointer and sets *errno* to [EACCES].

21(A) When the file exists and the requested read and/or write permission is denied, then a call to *fopen*(*filename*, *type*) returns a **NULL** pointer and sets *errno* to [EACCES].

Testing Requirements:
Test for instances, when *type* is "w", "w+", "wb", "wb+", or "w+b", that the file is not truncated."

22(A) When the file does not exist, and when write permission is denied for the parent directory of the file to be created, then a call to *fopen*(*filename*, *type*) returns a **NULL** pointer, sets *errno* to [EACCES], and the file is not created.

23(A) When *fopen*() is interrupted by a signal, then the call to *fopen*(*filename*, *type*) returns a **NULL** pointer, sets *errno* to [EINTR], and does not mark for update *st_ctime* and *st_mtime* fields of the file and parent directory and the *st_atime* field of the file.

24(A) When the named file is a directory, and when *type* is "w", "a", "r+", "w+", "a+", "wb", "ab", "r+b", "rb+", "w+b", "wb+", "a+b", or "ab+", then a call to *fopen*(*filename*, *type*) returns a **NULL** pointer, sets *errno* to [EISDIR], and does not mark for update *st_ctime*, *st_mtime*, and *st_atime* fields of the file.

25(A) If {OPEN_MAX}≤{PCTS_OPEN_MAX}:
When {OPEN_MAX} file descriptors have been opened, then a subsequent call to *fopen*(*filename*, *type*, *mode*) returns a **NULL** pointer ans sets *errno* to [EMFILE].
Testing Requirements:
Test for instances
— When the file does not exist and *type* is "w", "a", "w+", "a+", "wb", "ab", "wb+", "ab+", "w+b", or "a+b", that the file is not created, and
— When the file exists and *type* is "w", "w+", "wb", "wb+", or "w+b", that the file is not truncated

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Otherwise:

{PCTS_OPEN_MAX} files can be opened.

26({NAME_MAX} ≤ {PCTS_NAME_MAX}?C:UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the underlying file:

When the length of a pathname component of *filename* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *fopen(filename, type)* returns a value of (*int*)-1 and sets *errno* to [ENAMETOOLONG].

27({NAME_MAX}>{PCTS_NAME_MAX}?C :UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the underlying file:

When the length of a pathname component of *filename* equals {PCTS_NAME_MAX}, then a call to *fopen(filename, type)* succeeds.

28(A) If {PATH_MAX} ≤ {PCTS_PATH_MAX}:

When the length of the *filename* argument exceeds the maximum number of bytes in a pathname {PATH_MAX}, then a call to *fopen(filename, type)* returns a **NULL** pointer and sets *errno* to [ENAMETOOLONG].

Otherwise:

A filename of length {PCTS_PATH_MAX} can be opened.

29(B) When too many files are currently open in the system, then a call to *fopen(filename, type)* returns a **NULL** pointer and sets *errno* to [ENFILE].

Testing Requirements:

Test for instances

— When the file does not exist and *type* is "w", "a", "w+", "a+", "wb", "ab", "wb+", "ab+", "w+b", or "a+b", that the file is not created, and

— When the file exists and *type* is "w", "w+", "wb", "wb+", or "w+b", that the file is not truncated

See Reason 1 in Section 5. of POSIX.3 {4}.

30(A) When *type* is "r", "r+", "rb", "r+b", or "rb+", and when the named file does not exist, then a call to *fopen(filename, type)* returns a **NULL** pointer and sets *errno* to [ENOENT].

31(A) When a component of the path prefix of *filename* does not exist or *filename* points to an empty string, a call to *fopen(filename, type)* returns a **NULL** pointer and sets *errno* to [ENOENT].

32(B) When the directory or file system that would contain the new file cannot be extended, then a call to *fopen(filename, type)* returns a **NULL** pointer, sets *errno* to [ENOSPC], and the file is not created.

See Reason 1 in Section 5. of POSIX.3 {4}.

33(A) When a component of the *filename* prefix is not a directory, then a call to *fopen(filename, type)* returns a **NULL** pointer and sets *errno* to [ENOTDIR].

34(C) If the implementation supports a read-only file system:

When the named file resides on a read-only file system, and when *type* is "w", "a", "r+", "w+", "a+", "wb", "ab", "r+b", "rb+", "w+b", "wb+", "a+b", or "ab+", then a call to *fopen(filename, type)* returns a **NULL** pointer and sets *errno* to [EROFS].

Testing Requirements:

Test for instances

— When the file does not exist and *type* is "w", "a", "w+", "a+", "wb", "ab", "wb+", "ab+", "w+b", "a+b", that the file is not created, and

— When the file exists and *type* is "w", "w+", "wb", "wb+", or "w+b", that the file is not truncated

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.15 *fputc*() (8.1)

8.1.15.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype `int fputc(int, FILE *)` is declared. (See GA36 in 2.7.3)

Otherwise:

When the header <stdio.h> is included, then the function *fputc*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *fputc*() is defined as a macro when the header <stdio.h> is included:

When the macro *fputc*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *fputc*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.15.2 Description (8.2.3.6)

04(A) On a call to *fputc*() the *st_ctime* and *st_mtime* fields of the file are marked for update when data is caused to be written.

Testing Requirements:

Test for the cases when a call to *fputc*() is followed by calls to *fflush*(), *fclose*(), *exit*(), and *abort*().

8.1.15.3 Returns (8.2.3.11)

R01 When a call to *fputc*() completes unsuccessfully, then a value of EOF is returned and sets *errno* to indicate the error. (See Assertions 5-11 in 8.1.15.4)

8.1.15.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient capacity to accept the buffered data, then a call to *fputc*() returns a value of EOF and sets *errno* to [EAGAIN].

Testing Requirements:

Test for both a pipe and a FIFO.

06(A) When the stream pointer argument addresses a file descriptor that is not open for writing, then a call to *fputc*() returns a value of EOF and sets *errno* to [EBADF].

07(A) When *fputc*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *fputc*() returns a value of EOF and sets *errno* to [EINTR].

08(B) When attempting to write to a file that exceeds an implementation-defined maximum file size, then a call to *fputc*() returns a value of EOF and sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX.3 {4}.

09(B) When attempting to write to a device that has no more space for data, then a call to *fputc*() returns a value of EOF and sets *errno* to [ENOSPC].

See Reason 1 in Section 5. of POSIX.3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

10(A) When attempting to write to a pipe or a FIFO that is not open for reading by any process, then a call to *fputc*() returns a value of EOF, sets *errno* to [EPIPE]. and sends a SIGPIPE signal to the calling process.

Testing Requirements:
Test for both a pipe and a FIFO.

11(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, when TOSTOP is set, and when the *stream* referenced is the controlling terminal, then a call to *fputc*(*stream*) returns a value of EOF and sets *errno* to [EIO].

8.1.16 *fputs*() (8.1)

8.1.16.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype
`int fputs(const char *, FILE *)` is declared. (See GA36 in 2.7.3)

Otherwise:
When the header <stdio.h> is included, then the function *fputs*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3)

02(C) If *fputs*() is defined as a macro when th header <stdio.h> is included:
When the macro *fputs*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3)

03(C) If *fputs*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4)

8.1.16.2 Description (8.2.3.6)

04(A) On a call to *fputs*(), the *st_ctime* and *st_mtime* fields of the file are marked for update between its successful completion and the nest successful completion of a call to either *fflush*() or *fclose*() on the same stream or a call to *exit*() or *abort*().

8.1.16.3 Returns (8.2.3.11)

R01 When a call to *fputs*() completes unsuccessfully, then a value of EOF is returned and sets *errno* to indicate the error. (See Assertions 5-11 in 8.1.16.4)

8.1.16.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient capacity to accept the buffered data, then a call to *fputs*() returns a value of EOF and sets *errno* to [EAGAIN].

Testing Requirements:
Test for both a pipe and a FIFO.

06(A) When the stream pointer argument addresses a file descriptor that is not open for writing, then a call to *fputs*() returns a value of EOF and sets *errno* to [EBADF]

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

07(A) When *fputs*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *fputs*() returns a value of EOF and sets *errno* to [EINTR].

08(B) When attempting to write to a file that exceeds an implementation defined maximum file size, then a call to *fputs* () returns a value of EOF adn sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX.3 {4}.

09(B) When attempting to write to a device that has no more space for data, then a call to *fputs*() returns a value of EOF and sets *errno* to [ENOSPC].

See Reason 1 in Section 5. of POSIX.3 {4}.

10(A) When attempting to write to a pipe or a FIFO that is not open for reading by any process, then a call to *fputs*() returns a value of EOF, sets *errno* to [EPIPE], and sends a SIGPIPE signal to the calling process.

Testing Requirements:

Test for both a pipe and a FIFO.

11(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, when TOSTOP is set, and when the stream referenced is the controlling terminal, then a call to *fputs*(*s*, *stream*) returns a value of EOF and sets *errno* to [EIO].

8.1.17 *fread*() (8.1)

8.1.17.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype `size_t fread(void *,size_t, size_t, FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the headr <stdio.h> is included, then the function *fread*() is declared with the result type *size_t*. (See GA36 in 2.7.3.)

02(C) If *fread*() is defined as a macro when the header <stdio.h> is included:

When the macro *fread*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *size_t*. (See GA37 in 2.7.3)

03(C) If *fread*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4)

8.1.17.2 Description (8.2.3.5)

04(A) On a call to *fread*() when data was not supplied by a prior call to *ungetc*(), and when there has been no previous buffered read operation on the stream, then the *st_atime* field of the underlying file is marked for update.

8.1.17.3 Returns (8.2.3.11)

R01 When a call to *fread*() completes unsuccessfully, then the count returned is less than requested and sets *errno* to indicate the error. (See Assertions 5-9 in 8.1.17.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.17.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient data to process a read request, then a call to *fread*() returns a value less than requested and sets *errno* to [EAGAIN].

Testing Requirements:
Test for both a pipe and a FIFO.

06(A) When the stream pointer argument addresses a file descriptor that is not open for reading, then a call to *fread*() returns a value of (*size_t*)0 and sets *errno* to [EBADF].

07(A) When *fread*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *fread*() returns a value of (*size_t*)0 and sets *errno* to [EINTR].

08(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is ignoring or blocking the SIGTTIN signal and is a member of a background process group, and when the *stream* argument references the controlling terminal, then a call to *fread*(*ptr*, *size*, *nmemb*, *stream*) returns a value of (*size_t*)0 and sets *errno* to [EIO].

09(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is a member of an orphaned background process group, and when the *stream* argument references the controlling terminal, then a call to *fread*(*ptr*, *size*, *nmemb*, *stream*) returns a value of (*size_t*)0 and sets *errno* to [EIO].

8.1.18 *freopen*() (8.1)

8.1.18.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype
`FILE * freopen(const char *, const char *, FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *freopen*() is declared with the result type *FILE**. (See GA36 in 2.7.3.)

02(C) If *freopen*() is defined as a macro when the header <stdio.h> is included:
When the macro *freopen*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *FILE **. (See GA37 in 2.7.3.)

03(C) If *freopen*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.18.2 Description (8.2.3.3)

04(A) When the first filename component of the *filename* argument is ".", and the pathname does not begin with a slash, then *freopen*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(A) When the *filename* argument points to the string "/", then *freopen*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

06(A) When the *filename* argument points to the string "///", then *freopen*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

07(A) When the *filename* argument points to a string beginning with a single slash or beginning with three or more slashes, then *freopen*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *filename* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *freopen*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *filename* argument points to the string "F1/" and `F1` is a directory, then *freopen*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

10(A) When the argument *filename* points to the string "F1//" and `F1` is a directory, then *freopen*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *filename* argument points to the string "F1/F2", then *freopen*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the pathname argument points to the string "F1/./F2", then *freopen*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *filename* argument points to the string "F1/../F1/F2", then *freopen*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *filename* argument points to the string "F1//F2", then *freopen*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:

When the *filename* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *freopen*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *freopen*(*filename*, *mode*, *stream*), the pathname *filename* supports *filenames* containing any of the following characters:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789._-

(See GA02 in 2.2.2.32.)

17(A) On a call to *freopen*(*filename*, *mode*, *stream*), the pathname *filename* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) A call to *freopen*() allocates a file descriptor as a call to *open*() does.

19(A) The *freopen*() function deallocates the file descriptor underlying the stream indicated by its argument (i.e., this file descriptor will be made available for subsequent *open*()s by the process).

20(A) When a call to *freopen*() on the stream is performed, then all outstanding record locks owned by the process on the file associated with the stream are removed.

R01 When a call to *freopen*() is interrupted by a signal that is to be caught, then the call to *freopen*() returns a value of (*int*).1 and sets *errno* to [EINTR]. (See Assertion 32 in 8.1.18.4.)

21(B) When all file descriptors associated with a pipe have been closed by *freopen*(), then data remaining in the pipe is discarded.

See Reason 3 in Section 5. of POSIX.3 [4].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

22(A) When all file descriptors associated with a FIFO special file have been closed by *freopen*(), then data remaining in the FIFO is discarded.

23(B) When all file descriptors associated with an open file description have been closed by *freopen*(), then the open file description is freed.

See Reason 1 in Section 5. of POSIX.3 {4}.

24(B) When all file descriptors associated with a file have been closed by *freopen*(), and when the link count of the file is zero, then the space occupied by the file is freed.

See Reason 1 in Section 5. of POSIX.3 {4}.

25(B) When all file descriptors associated with a file have been closed by *freopen*(), and when the link count of the file is zero, then the file is no longer accessible.

See Reason 3 in Section 5. of POSIX.3 {4}.

26(A) When a file is closed via *freopen*() by the last process that had it open, all time-related fields still marked for update are updated.

27(A) When the referenced stream is writable, and when buffered data has not yet been written, then a call to *freopen(filename, mode, stream)* marks for update the *st_ctime* and *st_mtime* fields of the underlying file.

28(A) When a call to *freopen*() creates a file, then the file permission bits are set to allow both reading and writing for owner, for group and for other users except for those bits set in the file mode creation mask of the process. No execute (search) permission bits are set.

8.1.18.3 Returns (8.2.3.11)

R02 When a call to *freopen*() completes unsuccessfully, then a **NULL** pointer is returned and sets *errno* to indicate the error. (See Assertions 29-33, 35, 38-43 in 8.1.18.4.)

8.1.18.4 Errors (8.2.3.11)

29(A) When search permission is denied on a component of the *filename* prefix, then a call to *freopen(filename, mode, stream)* returns a **NULL** pointer and sets *errno* to [EACCES].

30(A) When the file exists and the requested read and/or write permission is denied, then a call to *freopen(filename, mode, stream)* returns a **NULL** pointer and sets *errno* to [EACCES].

Testing Requirements:

Test for instances when *mode* is "w", "w+", "wb", "wb+", or "w+b"that the file is not truncated.

31(A) When the file does not exist, and when write permission is denied for the parent directory of the file to be created, then a call to *freopen(filename, mode, stream)* returns a **NULL** pointer, sets *errno* to [EACCES], and the file is not created.

32(A) When *freopen*() is interrupted by a signal, then the call to *freopen(filename, mode, stream)* returns a **NULL** pointer, sets *errno* to [EINTR], and does not mark for update *st_ctime* and *st_mtime* fields of the file and parent directory and the *st_atime* field of the file.

33(A) When the named file is a directory, and when *mode* is "w", "a", "r+", "w+", "a+", "wb", "ab", "r+b", "rb+", "w+b", "wb+", "a+b", or "ab+", then a call to *freopen(filename, mode, stream)* returns a **NULL**, pointer, sets *errno* to [EISDIR], and does not mark for update *st_ctime*, *st_mtime*, and *st_atime* fields of the file.

34(B) If {OPEN_MAX} ≤ {PCTS_OPEN_MAX}:

When {OPEN_MAX} file descriptors have been opened, then a subsequent call to *freopen(filename, mode, stream)* returns a **NULL** pointer and sets *errno* to [EMFILE].

Testing Requirements:

Test for instances

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860388 351 ■

— When the file does not exist and *mode* is "w", "a", "w+", "a+", "wb", "ab", "wb+", "ab+", "w+b", or "a+b", that the file is not created, and
— When the file exists and *mode* is "w", "w+" , "wb" , "wb+", or "w+b", that the file is not truncated

Otherwise:

{PCTS_OPEN_MAX} file descriptors can be opened.

See Reason 1 in Section 5. of POSIX.3 {4}.

35({NAME_MAX}≤{PCTS_NAME_MAX}?C: UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the underlying file:

When the length of a pathname component of *filename* is longer than the maximum number of bytes in a filename {NAME_MAX}, then a call to *freopen(filename, mode, stream)* returns a **NULL** pointer and sets *errno* to [ENAMETOOLONG].

36({NAME_MAX}>{PCTS_NAME_MAX}?C :UNTESTED)

If the behavior associated with {_POSIX_NO_TRUNC} is supported for the underlying file:

When the length of a pathname component of *filename* equals {PCTS_NAME_MAX}, then a call to *freopen(filename, mode, stream)* succeeds.

37(A) If {PATH_MAX} ≤ {PCTS_PATH_MAX}:

When the length of *filename* argument exceeds the maximum number of bytes in a pathnAme {PATH_MAX}, then a call to *freopen(filename, mode, stream)* returns a **NULL** pointer and sets *errno* to [ENAMETOOLONG].

Otherwise:

A filename of {PCTS_PATH_MAX} can be opened.

38(B) When too many files are open in the system at the point at which *freopen*() tries to allocate a new file descriptor, then a call to *freopen(filename, mode, stream)* returns a **NULL** pointer and sets *errno* to [ENFILE].

Testing Requirements:

Test for instances

— When the file does not exist and *mode* is "w", "a", "w+", "a+", "wb ", "ab", "wb+", "ab+", "w+b", or "a+b", that the file is not created, and
— When the file exists and *mode* is "w", "w+", "wb", "wb+", or "w+b", that the file is not truncated

See Reason 1 in Section 5. of POSIX.3 {4}.

39(A) When the *mode* is "r", "r+", "rb", "r+b", or "rb+", and when the named file does not exist, then a call to *freopen(filename, mode, stream)* returns a **NULL** pointer and sets *errno* to [ENOENT].

40(A) When a component of the path prefix of *filename* does not exist or *filename* points to an empty string, then a call to *freopen(filename, mode, stream)* returns a **NULL** pointer and sets *errno* to [ENOENT].

41(B) When the directory or file system that would contain the new file cannot be extended, then a call to *freopen(filename, mode, stream)* returns a **NULL** pointer, sets *errno* to [ENOSPC], and the file is not created.

See Reason 1 in Section 5. of POSIX.3 {4}.

42(A) When a component of the *filename* prefix is not a directory, then a call to *freopen(filename, mode, stream)* returns a **NULL** pointer and sets *errno* to [ENOTDIR].

43(C) If the implementation supports a read-only file system:

When the named file resides on a read-only file system, and when *mode* is "w", "a", "r+", "w+", "a+", "wb", "ab", "r+b", "rb+"", "w+b", "wb+", "a+b", or "ab+", then a call to *freopen(filename, mode, stream)* returns 8 **NULL** pointer and sets *errno* to [EROFS].

Testing Requirements:

Test for instances

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— When the file does not exist and *mode* is "w", "a", "w+", "a+", "wb", "ab", "wb+", "ab+", "w+b", or "a+b", that the file is not created, and
— When the file exists and *mode* is "w", "w+", "wb", "wb+", or "w+b", that the file is not truncated

8.1.19 *fseek*() (8.1)

8.1.19.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `int fseek(FILE *, long int, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *fseek*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *fseek*() is defined as a macro when the header <stdio.h> is included:
When the macro *fseek*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *fseek*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.19.2 Description (8.2.3.7)

R01 When the file is capable of seeking, and when the file offset has been set beyond the end of existing data in the file by a call to *fseek*() followed by a call to *write*(), then data read from gaps in the file (areas where no data has been written) appear as though bytes with the value zero had been written. (See Assertion 18 in 9.5.2.)

04(A) When the file is capable of seeking, and when buffered data written by *fseek*(), if any, does not extend the file, then a call to *fseek*() does not, by itself, extend the size of the file.

05(A) When the stream is writable and buffered data had not yet been written, then a call to *fseek*() marks for update the *st_ctime* and *st_mtime* fields of the underlying file.

06(A) When a call to *fflush*() was the most recent operation [excluding *ftell*()], then a call to *fseek*() adjusts the file offset in the underlying open file description to reflect the location specified by the *fseek*() call.

8.1.19.3 Returns (8.2.3.11)

R02 When a call to *fseek*() completes unsuccessfully, then a nonzero value is returned and *errno* is set to indicate the error. (See Assertions 7-11 in 8.1.19.4)

8.1.19.4 Errors (8.2.3.11)

07(A) When the *stream* pointer argument addresses a file descriptor that is closed, then the call to *fseek*(*stream*, *offset*, *whence*) sets *errno* to [EBADF] and returns a nonzero value.

08(D) If the implementation provides buffered streams:
When *fseek*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *fseek*() sets *errno* to [EINTR] and returns a nonzero value.

See Reason 3 in Section 5. of POSIX.3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860390 TOT ■

09(D) If the implementation provides buffered streams:
When buffered data had not yet been written, and when an attempt to write the data will exceed an implementation-defined maximum file size, then a call to *fseek*() sets *errno* to [EFBIG] and returns a nonzero value.

See Reason 3 in Section 5. of POSIX.3 {4}.

10(D) If the implementation provides buffered streams:
When buffered data had not yet been written, and when an attempt is made to write to a device that has no more space for data, then a call to *fseek*() sets *errno* to [ENOSPC] and returns a nonzero value.

See Reason 1 in Section 5. of POSiX.3 {4}.

11(A) When the underlying open file descripton references a pipe or FIFO, then a call to *fseek*() sets *errno* to [ESPIPE], returns a nonzero value, and the value of the file pointer is not changed.

Testing Requirements:
Test for both a pipe and a FIFO.

8.1.20 *ftell*() (8.1)

8.1.20.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `long int ftell(FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *ftell*() is declared with the result type *long int*. (See GA36 in 2.7.3.)

02(C) If *ftell*() is defined as a macro when the header <stdio.h> is included:
When the macro *ftell*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *long int*. (See GA37 in 2.7.3.)

03(C) If *ftell*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.20.2 Description (8.2.3.10)

04(A) The result of a call to *ftell*() after an *fflush*() is the same as the result before the *fflush*().

8.1.20.3 Returns (8.2.3.11)

R01 When a call to *ftell*() completes unsuccessfully, then a value of (*long*)-1 is returned and sets *errno* to indicate the error. (See Assertions 5 and 6 in 8.1.20.4.)

8.1.20.4 Errors (8.2.3.11)

05(A) When the stream pointer argument addresses a file descriptor that is closed, then the call to *ftell*() returns a value of (*long*)-1 and sets *errno* to [EBADF].

06(A) When the underlying open file description references a pipe or a FIFO, then the call to *ftell*() returns a value of (*long*)-1 and sets *errno* to [ESPIPE].

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Testing Requirements:
Test for both a pipe and a FIFO.

8.1.21 *fwrite*() (8.1)

8.1.21.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype
`size_t fwrite(const void *, size_t, size_t, FInE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *fwrite*() is declared with the result type *size_t*. (See GA36 in 2.7.3.)

02(C) If *fwrite*() is defined as a macro when the header <stdio.h> is included:
When the macro *fwrite*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *size_t*. (See GA37 in 2.7.3.)

03(C) If *fwrite*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.21.2 Description (8.2.3.6)

04(A) On a call to *fwrite*(), the *st_ctime*and *st_mtime*fields of the file are marked for update between its successful completion and the next successful completion of a call to either *fflush*() or *fclose*() on the same stream or a call to *exit*()or *abort*().

05({PIPE_BUF}≤{PCTS_PIPE_BUF}?A:UN TESTED)

When *fwrite*() writes greater than zero bytes but fewer than requested, then the error indicator for the stream is set.

8.1.21.3 Returns (8.2.3.11)

R01 When a call to *fwrite*() completes unsuccessfully, and when no bytes are written, then a value of zero is returned and *errno* is set to indicate the error. (See Assertions 6-12 in 8.1.21.4.)

8.1.21.4 Errors (8.2.3.11)

06(A) When the underlying file descriptor references a pipe era FIFO that has the O_NONBLOCK flag set and contains insufficient capacity to accept any of the buffered data, then a call to *fwrite*() sets *errno* to [EAGAIN]and returns a value of zero.

Testing Requirements:
Test for both a pipe and a FIFO.

07(A) When the stream pointer argument addresses a file descriptor that is not open for writing, then a call to *fwrite*() sets *errno* to [EBADF] and returns a value of zero.

08(A) When *fwrite*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *fwrite*() sets *errno* to [EINTR] and returns a value of zero.

09(B) When attempting to write to a file that exceeds an implementation-defined maximum file size, and when no bytes are written to the file, then a call to *fwrite*() sets *errno* to [EFBIG] and returns a value of zero.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860392 882 ■

See Reason 3 in Section 5. of POSIX.3 {4}.

10(B) When attempting to write to a device that has no more space for data, and when no bytes are written to the file, then a call to *fwrite*() sets *errno* to [ENOSPC] and returns a value of zero.

See Reason 1 in Section 5. of POSIX.3 {4}.

11(A) When attempting to write to a pipe or a FIFO that is not open for reading by any process, then a call to *fwrite*() sets *errno* to [EPIPE], returns a value of zero, and sends a SIGPIPE signal to the calling process.

Testing Requirements:
Test for both a pipe and a FIFO.

12(PCTS_GTI_DEVICE?C :UNTESTED)

If the behavior associated with {POSIX_JOB_CONTROL} is supported:
When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, when TOSTOP is set, and when the *stream*referenced is the controlling terminal, then a call to *fwrite*(*ptr, size, nmemb, stream*) sets *errno* to [EIO] and returns a value of zero.

8.1.22 *getc*() (8.1)

8.1.22.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `int getc(FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *getc*() is either declared with result type *int*or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *getc*() is defined as a macro when the header <stdio.h> is included:
When the macro *getc*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *getc*() is defined as a macro in the header <stdio.h>:
It evaluates its argument one or more times, fully protected by parentheses when necessary, and protects its results value with extra parentheses when necessary.

8.1.22.2 Description (8.2.3.5)

04(A) When data was not supplied by a prior call to *ungetc*(), and when there has been no previous buffered read operation on the stream, then a call to *getc*() marks for update the *st_atime*field of the underlying file.

8.1.22.3 Returns (8.2.3.11)

R01 When a call to *getc*() completes unsuccessfully, then a value of EOF is returned and *sets errno* to indicate the error. (See Assertions 5-9 in 8.1.22.4.)

8.1.22.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient data to process a read request, then a call to *getc*() returns a value of EOF and sets *errno* to [EAGAIN].

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Testing Requirements:

Test for both a pipe and a FIFO.

06(A) When the stream pointer argument addresses a file descriptor that is not open for reading, then a call to *getc*() returns a value of EOF and sets *errno* to [EBADF].

07(A) When *getc*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *getc*() returns a value of EOF and sets *errno* to [EINTR]

08(C) If the behavior associated with {POSIX_JOB_CONTROL}is supported:

When the process is ignoring or blocking the SIGTTIN signal and is a member of a background process group, and when the *stream*argument references the controlling terminal, then a call to *getc*(*stream*) returns a value of EOF and sets *errno* to [EIO].

09(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process is a member of an orphaned background process group, and when the *stream*argument references the controlling terminal, then a call to *getc*(*stream*) returns a value of EOF and sets *errno* to [EIO].

8.1.23 *getchar*() (8.1)

8.1.23.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype `int getchar(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *getchar*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *getchar*() is defined as a macro when the header <stdio.h> is included:

When the macro *getchar*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *getchar*() is defined as a macro in the header <stdio.h>:

It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.23.2 Description (8.2.3.5)

04(A) On a call to *getchar*() when data was not supplied by a prior call to *ungetc*(), and when there has been no previous buffered read operation on the stream, then the *st_atime* field of the underlying file is marked for update.

8.1.23.3 Returns (8.2.3.11)

R01 When a call to *getchar*() completes unsuccessfully, then a value of EOF is returned and sets *errno* to indicate the error. (See Assertions 5-9 in 8.1.23.4.)

8.1.23.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient data to process a read request, then a call to *getchar*() returns a value of EOF and sets *errno* to [EAGAIN[.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Testing Requirements:
Test for both a pipe and a FIFO.

06(A) When *stdin* addresses a file descriptor that is not open for reading, then a call to *getchar*() returns a value of ECF and sets *errno* to [EBADF].

07(A) When *getchar*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *getchar*() returns a value of EOF and sets *errno* to [EINTR].

08(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is ignoring or blocking the SIGTTIN signal, is a member of a background process group, and is attempting to read from its controlling terminal, then a call to *getchar*() returns a value of EOF and sets *errno* to [EIO].

09(PCTS_GTI_DEVICE?C :UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is a member of an orphaned background process group and is attempting to read from its controlling terminal, then a call to *getchar*() returns a value of EOF and sets *errno* to [EIO].

8.1.24 *gets*() (8.1)

8.1.24.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `char * gets(char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *gets*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

02(C) If *gets*() is defined as a macro when the header <stdio.h> is included:
When the macro *gets*() is invoked with the correct argument types(or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

03(C) If *gets*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.24.2 Description (8.2.3.5)

04(A) When data was not supplied by a prior call to *ungetc*(), and when there has been no previous buffered read operation on the stream, then a call to *gets*() marks for update the *st_atime* field of the underlying file.

8.1.24.3 Returns (8.2.3.11)

R01 When a call to *gets*() completes unsuccessfully, then a **NULL** pointer is returned and sets *errno* to indicate the error. (See Assertions 5-9 in 8.1.24.4.)

8.1.24.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has he O_NONBLOCK flag set and contains insufficient data to process a read request, then a call to *gets*() returns a **NULL** pointer and sets *errno* to [EAGAIN].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Testing Requirements:
Test for both a pipe and a FIFO.

06(A) When *stdin* addresses a file descriptor that is not open for reading, then a call to *gets*() returns a **NULL** pointer and sets *errno* to [EBADF].

07(A) When *gets*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *gets*() returns a **NULL** pointer and sets *errno* to [EINTR].

08(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is ignoring or blocking the SIGTTIN signal and is a member of a background process group, and when stdin references the controlling terminal, then a call to *gets*(*buf*) returns a **NULL** pointer and sets *errno* to [EIO].

09(PCTS_GTI_DEVICE?C:UNTESTED)
If the behavior associated with {_POSIX_ JOB_CONTROL} is supported:
When the process is a member of an orphaned background process group, and when stalin references the controlling terminal, then a call to *gets*(*buf*) returns a **NULL** pointer and sets *errno* to [EIO].

8.1.25 *perror*() (8.1)

8.1.25.1 Synopsis (8.1)

01(C) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype
`void perror(const char *)` is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *perror*() is contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(D) If the implementation provides C Standard {2} support and *perror*() is defined as a macro when the header <stdio.h> is included:
When the macro *perror*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)

See Reason 2 in Section 5. of POSIX.3 {4}.

03(C) If *perror*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 in 1.3.3.)

8.1.25.2 Description (8.2.3.8)

04(A) A call to *perror*() marks for update the *st_ctime* and *st_mtime* fields of the underlying file associated with the standard error stream at some time between its successful completion and the completion of *fflush*() or *fclose*() on stderr, or *exit*() or *abort*().

8.1.25.3 Returns (8.2.3.11)

There are no assertions specific to this subclause.

8.1.25.4 Errors (8.2.3.11)

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.26 *printf*() (8.1)

8.1.26.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype `int printf(const char *, ...)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *printf*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *printf*() is defined as a macro when the header <stdio.h> is included:

When the macro *printf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *printf*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.26.2 Description (8.2.3.6)

04(A) On a call to *printf*(), the *st_ctime* and *st_mtime* fields of the file are marked for update between its successful completion and the next successful completion of a call to either *fflush*() or *fclose*() on stdout or a call to *exit*() or *abort*().

8.1.26.3 Returns (8.2.3.11)

R01 When a call to *printf*() completes unsuccessfully, then a negative value is returned and sets *errno* to indicate the error. (See Assertions 5-11 in 8.1.26.4.)

8.1.26.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient capacity to accept the buffered data, then a call to *printf*() returns a negative value and sets *errno* to [EAGAIN].

Testing Requirements:

Test for both a pipe and a FIFO.

06(A) When *stdout* addresses a file descriptor that is not open for writing,then a call to *printf*() returns a negative value and sets *errno* to [EBADF].

07(A) When *printf*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *printf*() returns a negative value and sets *errno* to [EINTR].

08(B) When attempting to write to a file that exceeds an implementation-defined maximum file size, then a call to *printf*()returns a negative value and sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX.3 {4}.

09(B) When attempting to write to a device that has no more space for data, then a call to *printf*() returns a negative value and sets *errno* to [ENOSPC].

See Reason 1 in Section 5. of POSIX.3 {4}.

10(A) When attempting to write to a pipe or a FIFO that is not open for reading by any process, then a call to *printf*() returns a negative value, sets *errno* to [EPIPE], and sends a SIGPIPE signal to the calling process.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Testing Requirements:

Test for both a pipe and a FIFO.

11 (PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, when TOSTOP is set, and when stdout is the controlling terminal, then a call to *printf*() returns a negative value and sets *errno* to [EIO].

8.1.27 *fprintf*() (8.1)

8.1.27.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <stdio .h> is included, then the function prototype `int fprintf(FILE *, const char *, ...)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *fprintf*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *fprintf*() is defined as a macro when the header <stdio.h> is included:

When the macro *fprintf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *fprintf*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.27.2 Description (8.2.3.6)

04(A) On a call to *fprintf*(), the *st_ctime* and *st_mtime* fields of the file are marked for update between its successful completion and the next successful completion of a call to either *fflush*() or *fclose*() on the same stream or a call to *exit*() or *abort*().

8.1.27.3 Returns (8.2.3.11)

R01 When a call to *fprintf*() completes unsuccessfully, then a negative value is returned and sets *errno* to indicate the error. (See Assertions 5-11 in 8.1.27.4.)

8.1.27.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient capacity to accept the buffered data, then a call to *fprintf*() returns a negative value and sets *errno* to [EAGAIN].

Testing Requirements:

Test for both a pipe and a FIFO.

06(A) When the stream pointer argument addresses a file descriptor that is not open for writing, then a call to *fprintf*() returns a negative value and sets *errno* to [EBADF].

07(A) When *fprintf*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *fprintf*() returns a negative value and sets *errno* to [EINTR].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

08(B) When attempting to write to a file that exceeds an implementation-defined maximum file size, then a call to *fprintf*() returns a negative value and sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX.3 {4}.

09(B) When attempting to write to a device that has no more space for data, then a call to *fprintf*() returns a negative value and sets *errno* to [ENOSPC].

See Reason 1 in Section 5. of POSIX.3 {4}.

10(A) When attempting to write to a pipe or a FIFO that is not open for reading by any process, then a call to *fprintf*() returns a negative value, sets *errno* to [EPIPE], and sends a SIGPIPE signal to the calling process.

Testing Requirements:
Test for both a pipe and a FIFO.

11(PCTS_GTI_DEVICE?C :UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, when TOSTOP is set, and when the *stream* referenced is the controlling terminal, then a call to *fprintf(stream, format)* returns a negative value and sets *errno* to [EIO].

8.1.28 *sprintf*() (8.1)

8.1.28.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype
`int sprintf(char *, const char *, ...)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *sprintf*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *sprintf*() is defined as a macro when the header <stdio.h> is included:
When the macro *sprintf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sprintf*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.28.2 Description (8.1)

04(B) If the implementation provides C Standard {2} support:
Element *sprintf*()complies with the requirements of the C Standard {2}. (See GA63 in 8..)

Otherwise:
Element *sprintf*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8..)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.28.3 Returns (8.1)

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.28.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.29 *vprintf*() (8.2.3.6)

8.1.29.1 Synopsis (8.2.3.6)

01(C) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype
`int vprintf(const char *, va_list)` is declared. (See GA36 in 2.7.3.)

02(C) If the implementation provides Common-Usage C and *vprintf*() from the C Standard {2}:
When the header <stdio.h> is included, then the function *vprintf*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

03(C) If the implementation provides *vprintf*() from the C Standard {2} and *vprintf*() is defined as a macro when the header <stdio.h> is included:
When the macro *vprintf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

04(C) If the implementation provides *vprintf*() from the C Standard {2} and *vprintf*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.29.2 Description (8.2.3.6)

05(C) If the implementation provides *vprintf*() from the C Standard {2}.
On a call to *vprintf*(), the *st_ctime* and *st_mtime* fields of the file are marked for update between its successful completion and the next successful completion of a call to either *fflush*() or *fclose*() on stdout or a call to *exit*() or *abort*().

8.1.29.3 Returns (8.2.3.11)

R01 When a call to *vprintf*() completes unsuccessfully, then a negative value is returned and sets *errno* to indicate the error. (See Assertions 6-12 in 8.1.29.4.)

8.1.29.4 Errors (8.2.3.11)

06(C) If the implementation provides *vprintf*() from the C Standard {2}:
When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient capacity to accept the buffered data, then a call to *vprintf*() returns a negative value and sets *errno* to [EAGAIN].
Testing Requirements:
Test for both a pipe and a FIFO.

07(C) If the implementation provides *vprintf*() from the C Standard {2}:
When stdout addresses a file descriptor that is not open for writing, then a call to *vprintf*() returns a negative value and sets *errno* to [EBADF].

08(C) If the implementation provides *vprintf*() from the C Standard {2}:
When *vprintf*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *vprintf*() returns a negative value and sets *errno* to [EINTR].

09(D) If the implementation provides *vprintf*() from the C Standard {2}:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When attempting to write to a file that exceeds an implementation-defined maximum file size, then a call to *vprintf*() returns a negative value and sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX.3 {4}.

10(D) If the implementation provides *vprintf*() from the C Standard {2}:

When attempting to write to a device that has no more space for data, then a call to *vprintf*() returns a negative value and sets *errno* to [ENOSPC].

See Reason 1 in Section 5. of POSIX.3 {4}.

11(C) If the implementation provides *vprintf*() from the C Standard {2}:

When attempting to write to a pipe or a FIFO that is not open for reading by any process, then a call to *vprintf*() returns a negative value, sets *errno* to [EPIPE], and sends a SIGPIPE signal to the calling process.

Testing Requirements:

Test for both a pipe and a FIFO.

12(PCTS_GTI_DEVICE?C:UNTESTED)

If the implementation provides *vprintf*() from the C Standard {2} and the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, when TOSTOP is set, and when the *stream* referenced is the controlling terminal, then a call to *vprintf*(*stream, format*) returns a negative value and sets *errno* to [EIO].

8.1.30 *vfprintf*() (8.2.3.6)

8.1.30.1 Synopsis (8.2.3.6)

01(C) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype `int vfprintf(FILE *, const char *, va_list)` is declared. (See GA36 in 2.7.3.)

02(C) If the implementation provides Common-Usage C and *vfprintf*() from the C Standard {2}:

When the header <stdio.h> is included, then the function *vfprintf*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

03(C) If the implementation provides *vfprintf*() from the C Standard {2} and *vfprintf*() is defined as a macro when the header <stdio.h> is included:

When the macro *vfprintf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

04(C) If the implementation provides *vfprintf*() from the C Standard {2} and *vfprintf*() is defined as a macro in the header <stdio.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.30.2 Description (8.2.3.6)

05(C) If the implementation provides *vfprintf*() from the C Standard {2}:

On a call to *vfprintf*(), the *st_ctime* and *st_mtime* fields of the file are marked for update between its successful completion and the next successful completion of a call to either *fflush*() or *fclose*() on the same stream or a call to *exit*() or *abort*().

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.30.3 Returns (8.2.3.11)

R01 When a call to *vfprintf*() completes unsuccessfully, then a negative value is returned and sets *errno* to indicate the error. (See Assertions 6-12 in 8.1.30.4.)

8.1.30.4 Errors (8.2.3.11)

06(C) If the implementation provides *vfprintf*() from the C Standard {2}:
When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient capacity to accept the buffered data, then a call to *vfprintf*() returns a negative value and sets *errno* to [EAGAIN].
Testing Requirements:
Test for both a pipe and a FIFO.

07C If the implementation provides *vfprintf*() from the C Standard {2}:
When the stream pointer argument addresses a file descriptor that is not open for writing, then a call to *vfprintf*() returns a negative value and sets *errno* to [EBADF].

08(C) If the implementation provides *vfprintf*() from the C Standard {2}:
When *vfprintf*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *vfprintf*()returns a negative value and sets *errno* to [EINTR].

09(D) If the implementation provides *vfprintf*() from the C Standard {2}:
When attempting to write to a file that exceeds an implementation-defined maximum file size, then a call to *vfprintf*() returns a negative value and sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX.3 {4}.

10(D) If the implementation provides *vfprintf*() from the C Standard {2}:
When attempting to write to a device that has no more space for data, then a call to *vfprintf*() returns a negative value and sets *errno* to [ENOSPC].

See Reason 1 in Section 5. of POSIX.3 {4}.

11(C) If the implementation provides *vfprintf*() from the C Standard {2}:
When attempting to write to a pipe or a FIFO that is not open for reading by any process, then a call to *vfprintf*() returns a negative value, sets *errno* to [EPIPE], and sends a SIGPIPE signal to the calling process.
Testing Requirements:
Test for both a pipe and a FIFO.

12(PCTS_GTI_DEVICE?C:UNTESTED)

If the implementation provides *vfprintf*() from the C Standard {2} and the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, when TOSTOP is set, and when the stream referenced is the controlling terminal, then a call to *vfprintf(stream, format)* returns a negative value and sets *errno* to [EIO].

8.1.31 *putc*() (8.1)

8.1.31.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `int putc(int, FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the header <stdio.h> is included, then the function *putc*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *putc*() is defined as a macro when the header <stdio.h> is included:
When the macro *putc*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *putc*() is defined as a macro in the header <stdio.h>:
It evaluates its first argument only once, evaluates its second argument one or more times—both arguments fully protected by parentheses when necessary—and protects its results value with extra parentheses when necessary.

8.1.31.2 Description (8.2.3.6)

04(A) On a call to *putc*(), the *st_ctime* and *st_mtime* fields of the file are marked for update between its successful completion and the next successful completion of a call to either *fflush*() or *fclose*() on the same stream or a call to *exit*() or *abort*().

8.1.31.3 Returns (8.2.3.11)

R01 When a call to *putc*() completes unsuccessfully, then a value of EOF is returned and sets *errno* to indicate the error. (See Assertions 5-11 in 8.1.31.4.)

8.1.31.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient capacity to accept the buffered data, then a call to *putc*() returns a value of EOF and sets *errno* to [EAGAIN].

Testing Requirements:
Test for both a pipe and a FIFO.

06(A) When the stream pointer argument addresses a file descriptor that is not open for writing, then a call to *putc*() returns a value of EOF and sets *errno* to [EBADF].

07(A) When *putc*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *putc*() returns a value of EOF and sets *errno* to [EINTR].

08(B) When attempting to write to a file that exceeds an implementation-defined maximum file size, then a call to *putc*() returns a value of EOF and sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX.3 {4}.

09(B) When attempting to write to a device that has no more space for data, then a call to putc() returns a value of EOF and sets errno to [ENOSPC].

See Reason 1 in Section 5. of POSIX.3 {4}.

10(A) When attempting to write to a pipe or a FIFO that is not open for reading by any process, then a call to *putc*() returns a value of EOF, sets *errno* to [EPIPE], and sends a SIGPIPE signal to the calling process.

Testing Requirements:
Test for both a pipe and a FIFO.

11(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, when TOSTOP is set, and when the *stream* referenced is the recalling terminal, then a call to *putc*(*c, stream*) returns a value of EOF and sets *errno* to [EIO].

8.1.32 *putchar*() (8.1)

8.1.32.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `int putchar(int)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *putchar*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *putchar*() is defined as a macro when the header <stdio.h> is included:
When the macro *putchar*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *putchar*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.32.2 Description (8.2.3.6)

04(A) On a call to *putchar*(), the *st_ctime* and *st_mtime* fields of the file are marked for update between its successful completion and the next successful completion of a call to either *fflush*() or *fclose*() on stdout or a call to *exit*() or *abort*().

8.1.32.3 Returns (8.2.3.11)

R01 When a call to *putchar*() completes unsuccessfully, then a value of EOF is returned and sets *errno* to indicate the error. (See Assertions 5-11 in 8.1.32.4.)

8.1.32.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient capacity to accept the buffered data, then a call to *putchar*() returns a value of EOF and sets *errno* to [EAGAIN].

Testing Requirements:
Test for both a pipe and a FIFO.

06(A) When *stdout* addresses a file descriptor that is not open for writing, then a call to *putchar*() returns a value of EOF and sets *errno* to [EBADF].

07(A) When *putchar*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *putchar*() returns a value of EOF and sets *errno* to [EINTR].

08(B) When attempting to write to a file that exceeds an implementation-defined maximum file size, then a call to *putchar*() returns a value of EOF and sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX.3 {4}.

09(B) When attempting to write to a device that has no more space for data, then a call to *putchar*() returns a value of EOF and sets *errno* to [ENOSPC].

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

See Reason 1 in Section 5. of POSIX.3 {4}.

10(A) When attempting to write to a pipe or a FIFO that is not open for reading by any process, then a call to *putchar*() returns a value of EOF, sets *errno* to [EPIPE], and sends a SIGPIPE signal to the calling process.

Testing Requirements:
Test for both a pipe and a FIFO.

11(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, when TOSTOP is set, and when stdout is the controlling terminal, then a call to *putchar*() returns a value of EOF and sets *errno* to [EIO].

8.1.33 *puts*() (8.1)

8.1.33.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `int puts(const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *puts*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *puts*() is defined as a macro when the header <stdio.h> is included:
When the macro *puts*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *puts*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.33.2 Description (8.2.3.6)

04(A) On a call to *puts*(), the *st_ctime* and *st_mtime* fields of the file are marked for update between its successful completion and the next successful completion of a call to either *fflush*() or *fclose*() on stdout or a call to *exit*() or *abort*().

8.1.33.3 Returns (8.2.3.11)

R01 When a call to *puts*() completes unsuccessfully, then a value of EOF is returned and sets *errno* to indicate the error. (See Assertions 5-11 in 8.1.33.4.)

8.1.33.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient capacity to accept the buffered data, then a call to *puts*() returns a value of EOF and sets *errno* to [EAGAIN].

Testing Requirements:
Test for both a pipe and a FIFO.

06(A) When stdout addresses a file descriptor that is not open for writing, then a call to *puts*() returns a value of EOF and sets *errno* to [EBADF].

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

07(A) When *puts*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *puts*() returns a value of EOF and sets *errno* to [EINTR].

08(B) When attempting to write to a file that exceeds an implementation-defined maximum file size, then a call to *puts*() returns a value of EOF and sets *errno* to [EFBIG].

See Reason 3 in Section 5. of POSIX. 3 {4}.

09(B) When attempting to write to a device that has no more space for data, then a call to *puts*() returns a value of EOF and sets *errno* to [ENOSPC].

See Reason 1 in Section 5. of POSIX.3 {4}.

10(A) When attempting to write to a pipe or a FIFO that is not open for reading by any process, then a call to *puts*() returns a value of EOF, sets *errno* to [EPIPE], and sends a SIGPIPE signal to the calling process.

Testing Requirements:
Test for both a pipe and a FIFO.

11(PCTS_GTI_DEVICE?C:UNTESTED)

If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is neither ignoring nor blocking the SIGTTOU signal and is a member of an orphaned background process group, when TOSTOP is set, and when stdout is the controlling terminal, then a call to *puts*() returns a value of EOF and sets *errno* to [EIO].

8.1.34 *remove*() (8.1)

8.1.34.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `int remove(const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *remove*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *remove*() is defined as a macro when the header <stdio.h> is included:
When the macro *remove*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *remove*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.34.2 Description (8.1)

04(A) When the first filename component of the *filename* argument is ".", and the pathname does not begin with a slash, then *remove*() resolves the pathname by locating the second filename component (when specified) in the current working directory. (See GA14 in 2.3.6.)

05(B) When the *filename* argument points to the string "/", then *remove*() resolves the pathname to the root directory of the process. (See GA15 in 2.3.6.)

See Reason 1 in Section 5. of POSIX.3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

06(B) When the *filename* argument points to the string "///", then *remove*() resolves the pathname to the root directory of the process. (See GA16 in 2.3.6.)

See Reason 1 in Section 5. of POSIX.3 [4].

07(A) When the *filename* argument points to a string beginning with a single slash or beginning with three or more slashes, then *remove*() resolves the pathname by locating the first filename component of the pathname in the root directory of the process. (See GA17 in 2.3.6.)

08(A) When the first filename component of the *filename* argument is "..", the pathname does not begin with a slash, and the current working directory is not the root directory of the process, then *remove*() resolves the pathname by locating the second filename component (when specified) in the parent directory of the current working directory. (See GA18 in 2.3.6.)

09(A) When the *filename* argument points to the string "F1/" and `F1` is a directory, then *remove*() resolves the pathname to `F1`, which is in the current working directory. (See GA19 in 2.3.6.)

10(A) When the *filename* argument points to the string "F1//" and F1 is a directory, then *remove*() resolves the pathname to `F1`, which is in the current working directory. (See GA20 in 2.3.6.)

11(A) When the *filename* argument points to the string "F1/F2", then *remove*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA21 in 2.3.6.)

12(A) When the pathname argument points to the string "F1/./F2", then *remove*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA22 in 2.3.6.)

13(A) When the *filename* argument points to the string "F1/../F1/F2", then *remove*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA23 in 2.3.6.)

14(A) When the *filename* argument points to the string "F1//F2", then *remove*() resolves the pathname to the file `F2` in the directory `F1`, which is in the current working directory. (See GA24 in 2.3.6.)

15(C) If {_POSIX_NO_TRUNC} is not supported in the corresponding directory:
When the *filename* component is a string of more than {NAME_MAX} bytes in a directory for which {_POSIX_NO_TRUNC} is not supported, then *remove*() resolves the pathname component by truncating it to {NAME_MAX} bytes. (See GA25 in 2.3.6.)

16(A) On a call to *remove(filename)*, the pathname *filename* supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

17(A) On a call to *remove(filename)*, the pathname *filename* retains the unique identity of its upper- and lowercase letters. (See GA03 in 2.2.2.32.)

18(A) A call to *remove*() marks for update the *st_ctime* and the *st_mtime* field of the parent directory of the file.

19(A) When the link count of the file is not zero, then a call to *remove*() marks for update the *st_ctime* field of the file.

8.1.34.3 Returns (8.2.3.11)

There are no assertions specific to this subclause.

8.1.34.4 Errors (8.2.3.11)

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.35 *rename*() (8.1)

8.1.35.1 Synopsis (8.1)

The assertions for this element are identical with the assertions as specified in 5.5.3.1 through 5.5.3.4. Therefore, these assertions are all referenced assertions.

R01-R69 (See Assertions 1-69 in 5.5.3.1 through 5.5.3.4.)

There are no assertions specific to this subclause.

8.1.36 *rewind*() (8.1)

8.1.36.1 Synopsis (8.1)

01(C) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `void rewind(FILE *)` is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *rewind*() is contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(D) If the implementation provides C Standard {2} support and *rewind*() is defined as a macro when the header <stdio.h> is included:
When the macro *rewind*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)

See Reason 2 in Section 5. of POSIX.3 {4}.

03(C) If *rewind*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 in 1.3.3.)

8.1.36.2 Description (8.2.3.7)

04(A) When the stream is writable and buffered data has not yet been written, then a call to *rewind*() marks for update the *st_ctime* and *st_mtime* fields of the underlying file.

8.1.36.3 Returns (8.2.3.11)

There are no assertions specific to this subclause.

8.1.36.4 Errors (8.2.3.11)

There are no assertions specific to this subclause.

8.1.37 *scanf*() (8.1)

8.1.37.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `int scanf(const char *, ...)` is declared. (See GA36 in 2.7.3.)

Otherwise:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the header <stdio.h> is included, then the function *scanf*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *scanf*() is defined as a macro when the header <stdio.h> is included:
When the macro *scanf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *scanf*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.37.2 Description (8.2.3.5)

04(A) On a call to *scanf*() when data was not supplied by a prior call to *ungetc*(), and when there has been no previous buffered read operation on the stream, then the *st_atime* field of the underlying file is marked for update.

8.1.37.3 Returns (8.2.3.11)

R01 When a call to *scanf*() completes unsuccessfully, then a value of EOF is returned and sets *errno* to indicate the error. (See Assertions 5-9 in 8.1.37.4.)

8.1.37.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient data to process a read request, then a call to *scanf*() returns a value of EOF and sets *errno* to [EAGAIN].

Testing Requirements:
Test for both a pipe and a FIFO.

06(A) When *stadin* addresses a file descriptor that is not open for reading, then a call to *scanf*() returns a value of EOF and sets *errno* to [EBADF].

07(A) When *scanf*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *scanf*() returns a value of EOF and sets *errno* to [EINTR].

08(C) If the behavior associated with {_.POSIX_JOB_CONTROL} is supported:
When the process is ignoring or blocking the SIGTTIN signal and is a member of a background process group, and when stdin references the controlling terminal, then a call to *scanf*() returns a value of EOF and sets *errno* to [EIO].

09(C) If the behavior associated with [_POSIX_JOB_CONTROL] is supported:
When the process is a member of an orphaned background process group, and when stdin is the controlling terminal, then a call to *scanf*() returns a value of EOF and sets *errno* to [EIO].

8.1.38 *fscanf*() (8.1)

8.1.38.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype
`int fscanf(FILE *, const char *, ...)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *fscanf*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

02(C) If *fscanf*() is defined as a macro when the header <stdio.h> is included:
When the macro *fscanf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *fscanf*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.38.2 Description (8.2.3.5)

04(A) On a call to *fscanf*() when data was not supplied by a prior call to *ungetc*(), and when there has been no previous buffered read operation on the stream, then the *st_atime* field of the underlying file is marked for update.

8.1.38.3 Returns (8.2.3.11)

R01 When a call to *fscanf*() completes unsuccessfully, then a value of EOF is returned and sets *errno* to indicate the error. (See Assertions 5-9 in 8.1.38.4.)

8.1.38.4 Errors (8.2.3.11)

05(A) When the underlying file descriptor references a pipe or a FIFO that has the O_NONBLOCK flag set and contains insufficient data to process a read request, then a call to *fscanf*() returns a value of EOF and sets *errno* to [EAGAIN].

Testing Requirements:
Test for both a pipe and a FIFO.

06(A) When the stream pointer argument addresses a file descriptor that is not open for reading, then a call to *fscanf*() returns a value of EOF and sets *errno* to [EBADF].

07(A) When *fscanf*() is terminated due to receipt of a signal, and when no data was transferred, then the call to *fscanf*() returns a value of EOF and sets *errno* to [EINTR].

08(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is ignoring or blocking the SIGTTIN signal and is a member of a background process group, and when the *stream* argument references the controlling terminal, then a call to *fscanf*(*stream*, *format*) returns a value of EOF and sets *errno* to [EIO].

09(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the process is a member of an orphaned background process group, and when the *stream* argument references the controlling terminal, then a call to *fscanf*(*stream*, *format*) returns a value of EOF and sets *errno* to [EIO].

8.1.39 *sscanf*() (8.1)

8.1.39.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype
`int sscanf(const char *, const char *, ...)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *sscanf*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *sscanf*() is defined as a macro when the header <stdio.h> is included:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860410 628 ■

When the macro *sscanf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sscanf*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.39.2 Description (8.1)

04(B) If the implementation provides C Standard {2} support:
Element *sscanf*() complies with the requirements of the C Standard {2}. (See GA63 in 8.)

Otherwise:
Element *sscanf*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8.)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.39.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.39.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.40 *setbuf*() (8.1)

8.1.40.1 Synopsis (8.1)

01(C) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype
`void setbuf(FILE *, char *)` is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *setbuf*() is contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(D) If the implementation provides C Standard {2} support and *setbuf*() is defined as a macro when the header <stdio.h> is included:
When the macro *setbuf*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)

See Reason 2 in Section 5. of POSIX.3 {4}.

03(C) If *setbuf*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 in 1.3.3.)

8.1.40.2 Description (8.1)

04(B) If the implementation provides C Standard {2} support:
Element *setbuf*() complies with the requirements of the C Standard {2}. (See GA63 in 8.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Otherwise:

Element *setbuf*()complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8.)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.40.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.40.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.41 *tmpfile*() (8.1)

8.1.41.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype `FILE * tmpfile(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *tmpfile*() is declared with the result type *FILE* *. (See GA36 in 2.7.3.)

02(C) If *tmpfile*() is defined as a macro when the header <stdio.h> is included:

When the macro *tmpfile*() is invoked, then it expands to an expression with the result type *FILE* *. (See GA37 in 2.7.3.)

03(C) If *tmpfile*() is defined as a macro in the header <stdio.h>:

It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.41.2 Description (8.2.3.9)

04(A) A call to *tmpfile*() allocates a file descriptor as a call to *open*() does.

05(B) When a call to *tmpfile*() creates a file, then the file permission bits are set to allow both reading and writing for owner, for group, and for other users except for those bits set in the file mode creation mask of the process. No execute (search) permission bits are set.

See Reason 1 in Section 5. of POSIX.3 {4}.

8.1.41.3 Returns (8.2.3.11)

R01 When a call to *tmpfile*() completes unsuccessfully, then a **NULL** pointer is returned and sets *errno* to indicate the error. (See Assertions 6 and 7 in 8.1.41.4.)

8.1.41.4 Errors (8.2.3.11)

06(A) If {OPEN_MAX} ≤ {PCTS_OPEN_MAX}:

When {OPEN_MAX} files have been opened, then a subsequent call to *tmpfile*() returns a **NULL** pointer and sets *errno* to [EMFILE].

Otherwise:

{PCTS_ OPEN_MAX} files can be opened.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

07(B) When too many files are currently open in the system, a call to *tmpfile*() returns a **NULL** pointer and sets *errno* to [ENFILE].

See Reason 1 in Section 5. of POSIX.3 {4}.

8.1.42 *tmpnam*() (8.1)

8.1.42.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `char * tmpnam(char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *tmpnam*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

02(C) If *tmpnam*() is defined as a macro when the header <stdio.h> is included:
When the macro *tmpnam*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

03(C) If *tmpnam*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.42.2 Description (8.1)

04(B) If the implementation provides C Standard {2} support:
Element *tmpnam*() complies with the requirements of the C Standard {2}. (See GA63 in 8.)

Otherwise:
Element *trnpnam*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8.)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.42.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.42.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.43 *ungetc*() (8.1)

8.1.43.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `int ungetc(int, FILE *)` is declared. (SeeGA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *ungetc*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *ungetc*() is defined as a macro when the header <stdio.h> is included:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the macro *ungetc*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *ungetc*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.43.2 Description (8.1)

04(B) If the implementation provides C Standard {2} support:
Element *ungetc*() complies with the requirements of the C Standard {2}. (See GA63 in 8.)

Otherwise:
Element *ungetc*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8.)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.43.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.43.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.44 *abs*() (8.1)

8.1.44.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdlib.h> is included, then the function prototype `int abs(int)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdlib.h> is included, then the function *abs*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *abs*() is defined as a macro when the header <stdlib.h> is included:
When the macro *abs*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *abs*() is defined as a macro in the header <stdlib.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.44.2 Description (8.1)

04(B) If the implementation provides C Standard {2} support:
Element *abs*()complies with the requirements of the C Standard {2}. (See GA63 in 8..)

Otherwise:
Element *abs*() complies with the requirements of the C Standard {2} unless Amended by the PCD.1. (See GA63 in 8.)

See Reason 2 in Section 5. of POSIX.3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.44.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.44.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.45 *atof*(), *atoi*(), *atol*() (8.1)

8.1.45.1 Synopsis (8.1)

01(A) For *atof*():
If the implementation provides C Standard {2} support:
When the header <stdlib.h> is included, then the function prototype `double atof(const char *)` is declared. (See GA36 in 2.7.3.)
Otherwise
When the header <stdlib.h> is included, then the function *atof*() is declared with the result type *double*. (See GA36 in 2.7.3.)

For *atoi*():
If the implementation provides C Standard {2} support:
When the header <stdlib.h> is included, then the function prototype `int atoil(const char *r)` is declared. (See GA36 in 2.7.3.)
Otherwise:
When the header <stdlib.h> is included, then the function *atoi*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

For *atol*():
If the implementation provides C Standard {2} support:
When the header <stdlib.h> is included, then the function prototype `long atol(const char *)` is declared. (See GA36 in 2.7.3.)
Otherwise:
When the header <stdlib.h> is included, then the function *atol*() is declared with the result type *long*. (See GA36 in 2.7.3.)

02(C) For *atof*():
If *atof*() is defined as a macro when the header <stdlib.h> is included:
When the macro *atof*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type double. (See GA37 in 2.7.3.)

For *atoi*():
If *atoi*() is defined as a macro when the header <stdlib.h> is included:
When the macro *atoi*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

For *atol*():
If *atol*() is defined as a macro when the header <stdlib.h> is included:
When the macro *atol*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *long*. (See GA37 in 2.7.3.)

03(C) If the *atof*(), *atoi*(), or *atol*() function is defined as a macro in the header <stdlib.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.45.2 Description (8.1)

04(B) For *funct*() of *atof*(), *atoi*(), and *atol*():
If the implementation provides C Standard {2} support:
Element *funct*() complies with the requirements of the C Standard {2}. (See GA63 in 8..)
Otherwise:
Element *funct*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8..)
See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.45.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.45.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.46 *rand*() (8.1)

8.1.46.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:
When the header <stdlib.h> is included, then the function proto-type `int rand(void)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdlib.h> is included, then the function *rand*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *rand*() is defined as a macro when the header <stdlib.h> is included:
When the macro *rand*() is invoked, then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *rand*() is defined as a macro in the header <stdlib.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.46.2 Description (8.1)

04(B) If the implementation provides C Standard {2} support:
Element *rand*() complies with the requirements of the C Standard {2}. (See GA63 in 8..)

Otherwise:
Element *rand*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8.)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.46.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.46.4 Errors (8.1)

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.47 *srand*() (8.1)

8.1.47.1 Synopsis (8.1)

01(C) If the implementation provides C Standard {2} supports:
When the header <stdlib.h> is included, then the function proto-type void standlparunsigned intrpar is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *srand*() is contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(D) If the implementation provides C Standard {2} support and *srand*() is defined as a macro when the header <stdlib.h> is included:
When the macro *srand*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)

See Reason 2 in Section 5. of POSIX.3 {4}.

03(C) If *srand*() is defined as a macro in the header <stdlib.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 in 1.3.3.)

8.1.47.2 Description (8.1)

04(B) If the implementation provides C Standard {2} support:
Element *srand*() complies with the requirements of the C Standard {2}. (See GA63 in 8..)

Otherwise:
Element *srand*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8..)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.47.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.47.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.48 *calloc*(), *free*(), *malloc*(), *realloc*() (8.1)

8.1.48.1 Synopsis (8.1)

01(A) For *calloc*():
If the implementation provides C Standard {2} support:
When the header <stdlib.h> is included, then the function prototype
`void * calloc(size_t, size_t)` is declared. (See GA36 in 2.7.3.)

For *free*():
If the implementation provides C Standard {2} support:
When the header <stdlib.h> is included, then the function prototype
`void free(void *)` is declared. (See GA36 in 2.7.3.)

For *malloc*():
If the implementation provides C Standard {2} support:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the header <stdlib.h> is included, then the function prototype
`void * malloc(size_t)` is declared. (See GA36 in 2.7.3.)

For *realloc*():
If the implementation provides C Standard {2} support:
When the header <stdlib.h> is included, then the function prototype
`void * realloc(void *, size_t)` is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for functions *calloc*(), *free*(), *malloc*(), and *realloc*() are contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(C) For *calloc*():
If the implementation provides C Standard {2} support and *calloc*() is defined as a macro when the header <stdlib.h> is included:
When the macro *calloc*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void* *. (See GA37 in 2.7.3.)

02(D) For *free*():
If the implementation provides C Standard {2} support and *free*() is defined as a macro when the header <stdlib.h> is included:
When the macro *free*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)
See Reason 2 in Section 5. of POSIX.3 {2}.

02(C) For *malloc*():
If the implementation provides C Standard {2} support and *malloc*() is defined as a macro when the header <stdlib.h> is included:
When the macro *malloc*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void* *. (See GA37 in 2.7.3.)

For *realloc*():
If the implementation provides C Standard {2} support and *realloc*() is defined as a macro when the header <stdlib.h> is included:
When the macro *realloc*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void* *. (See GA37 in 2.7.3.)

03(C) For *funct*() of *calloc*(), *malloc*(), and *realloc*():
If *funct*() is defined as a macro in the header <stdlib.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

For *free*():
If *free*() is defined as a macro in the header <stdlib.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 in 1.3.3.)

8.1.48.2 Description (8.1)

04(B) For *funct*() of *calloc*(), *free*(), *malloc*(), and *realloc*():
If the implementation provides C Standard {2} support:
Element *funct*() complies with the requirements of the C Standard {2}. (See GA63 in 8..)
Otherwise:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Element *funct*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8..)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.48.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.48.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.49 *abort*() (8.1)

8.1.49.1 Synopsis (8.1)

01(C) If the implementation provides C Standard {2} support:
When the header <stdlib.h> is included, then the function prototype `void abort(void)` is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *abort*() is contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(C) If the implementation provides C Standard {2} support and *abort*() is defined as a macro when the header <stdlib.h> is included:
When the macro *abort*() is invoked, then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)

8.1.49.2 Description (8.1)

03(A) A call to *abort*() that terminates the process with SIGABRT has the effect of a call to *fclose*() on every open stream.

04(A) When the SIGABRT signal is being ignored or blocked, or is being caught and the catching function returns, then a call to *abort*() that terminates the process with SIGABRT has the effect of a call to *fclose*() on every open *stream*.

05(A) When the SIGABRT signal is being ignored or blocked or the signal handler associated with SIGABRT returns, then on a call to *abort*() the process ceases the execution of subsequent program statements. (See GA40 in 3.2.)

06(A) When the SIGABRT signal is being ignored or blocked or the signal handler associated with SIGABRT returns, then on a call to *abort*() all open file descriptors are closed. (See GA41 in 3.2.)

07(B) When the SIGABRT signal is being ignored or blocked or the signal handler associated with SIGABRT returns, then on a call to *abort*() all open directory streams are closed. (See GA42 in 3.2.)

See Reason 2 in Section 5. of POSIX.3 {4}.

08(A) When the SIGABRT signal is being ignored or blocked or the signal handler associated with SIGABRT returns, and when the parent process of the calling process is waiting for this process, then on a call to *abort*() the parent process is notified of the calling process termination, and the low-order 8 b of status are made available to it. (See GA43 in 3.2.)

09(A) When the SIGABRT signal is being ignored or blocked or the signal handler associated with SIGABRT returns, and when the parent process of the calling process is not waiting for the process, then on a call to *abort*() the exit status code is saved for return to the parent process whenever the parent process executes a subsequent *wait*() or *waitpid*() for this process. (See GA44 in 3.2.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10(A) When the SIGABRT signal is being ignored or blocked or the signal handler associated with SIGABRT returns, and when a parent process, which is not a controlling process, terminates by calling *abort*() without waiting for its child processes to terminate, then the remaining child processes are not terminated and are assigned a new parent process ID. (See GA45 in 3.2.)

11(C) If either the behavior associated with {_POSIX_JOB_CONTROL} is supported or SIGCHLD is supported:
When a process terminates by calling *abort*() and when the SIGABRT signal is being ignored or blocked or the signal handler associated with SIGABRT returns, then a SIGCHLD signal is sent to its parent process. (See GA46 in 3.2.)

12(PCTS_GTI_DEVICE?A:UNTESTED)

When a controlling process terminates by calling *abort*(), and when the SIGABRT signal is being ignored or blocked or the signal handler associated with SIGABRT returns, then a SIGHUP signal is sent to each process in the foreground process group of the controlling terminal belonging to the controlling process. The remaining child processes, which are not terminated upon delivery of the SIGHUP signal, are assigned a new parent process ID. (See GA47 in 3.2.)

13(PCTS_GTI_DEVICE?A:UNTESTED)

When the SIGABRT signal is being ignored or blocked or the signal handler associated with SIGABRT returns , and when the calling process, which is a controlling process, terminates by calling *abort*(), then the controlling terminal associated with the session is disassociated from the session. (See GA48 in 3.2.)

14(C) If the behavior associated with {_POSIX_JOB_CONTROL} is supported:
When the SIGABRT signal is being ignored or blocked or the signal handler associated with SIGABRT returns, when termination of the process causes a process group to be orphaned, and when any member of the newly orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal is sent to each process in the newly orphaned process group. (See GA49 in 3.2.)

15(A) When the SIGABRT signal is being ignored or blocked or the signal handler associated with SIGABRT returns, then on a call to *abort*() the status made available to *wait*() or *waitpid*() is that of a process terminated by the SIGABRT signal. (See GA50 in 3.2.)

8.1.49.3 Returns (8.1)

R01 A call to *abort*() does not return. (See Assertion 5 in 8.1.49.2.)

8.1.49.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.50 *exit*() (8.1)

8.1.50.1 Synopsis (8.1)

01(C) If the implementation provides C Standard {2} support:
When the header <stdlib.h> is included, then the function prototype
`void exit(int)` is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *exit*() is contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(D) If the implementation provides C Standard {2} support and *exit*() is defined as a macro when the header <stdlib.h> is included:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the macro *exit*() is invoked with the correct argument types lparor compatible argument types in the case that C Standard {2} support is providedrpar, then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)

See Reason 2 in Section 5. of POSIX.3 {4}.

03(C) If *exit*() is defined as a macro in the header <stdlib.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 in 1.3.3.)

8.1.50.2 Description (8.1)

04(A) When <stdlib.h> is included, then EXIT_SUCCESS is defined as an integral expression that evaluates to a value of zero.

05(A) When <stdlib.h> is included, then EXIT_FAILURE is defined as an integral expression that evaluates to a nonzero value.

06(A) A call to *exit*() that terminates the process has the effect of a call to *fclose*() on every open *stream*.

07(A) If the function *atexit*() is supported:
When the process returns from the functions previously registered by *atexit*(), then the process ceases the execution of subsequent program statements. (See GA40 in 3.2.)

Otherwise:
The process ceases the execution of subsequent program statements. (See GA40 in 3.2.)

08(A) On a call to *exit*() that terminates the process, all open file descriptors are closed. (See GA41 in 3.2.)

09(B) On a call to *exit*() that terminates the process, all open directory streams are closed. (See GA42 in 3.2.)

See Reason 2 in Section 5. of POSIX.3 {4}.

10(A) If the function *atexit*() is supported:
When the process returns from the functions previously registered by *atexit*(), and when the parent process of the calling process is waiting for this process, then the parent process is notified of the calling process termination, and the low-order 8 b of status are made available to it. (See GA43 in 3.2.)

Otherwise:
When the process calls *exit*(), and when the parent process of the calling process is waiting for this process, then the parent process is notified of the calling process termination, and the low-order 8 b of status are made available to it. (See GA43 in 3.2.)

11(A) If the function *atexit*() is supported:
When the process returns from the functions previously registered by *atexit*(), and when the parent process of the calling process is not waiting for the process, then the exit status code is saved for return to the parent process whenever the parent process executes a subsequent *wait*() or *waitpid*() for this process. (See GA44 in 3.2.)

Otherwise:
When the process calls *exit*(), and when the parent process of the calling process is not waiting for the process, then the exit status code is saved for return to the parent process whenever the parent process executes a subsequent *wait*() or *waitpid*() for this process. (See GA44 in 3.2.)

12(A) If the function *atexit*() is supported:
When the process returns from the functions previously registered by *atexit*(), and when a parent process that is not a controlling process terminates without waiting for its child processes to terminate, then the remaining child processes are not terminated and are assigned a new parent process ID. (See GA45 in 3.2.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Otherwise:

When the process calls *exit*(), and when a parent process that is not a controlling process terminates without waiting for its child processes to terminate, then the remaining child processes are not terminated and are assigned a new parent process ID. (See GA45 in 3.2.)

13(C) If the function *atexit*() is supported and either the behavior associated with {_POSIX_JOB_CONTROL} is supported or SIGCHLD is supported:

When the process returns from the functions previously registered by *atexit*(), then a SIGCHLD signal is sent to its parent process. (See GA46 in 3.2.)

14(C) If the function *atexit*() is not supported and either the behavior associated with {_POSIX_JOB_CONTROL} is supported or SIGCHLD is supported:

When the process calls *exit*(), then a SIGCHLD signal is sent to its parent process. (See GA46 in 3.2.)

15(PCTS_GTI_DEVICE?A:UNTESTED)

If the function *atexit*() is supported:

When a controlling process returns from the functions previously registered by *atexit*(), then a SIGHUP signal is sent to each process in the foreground process group of the controlling terminal belonging to the controlling process. The remaining child processes, which are not terminated upon delivery of the SIGHUP signal, are assigned a new parent process ID. (See GA47 in 3.2.)

Otherwise:

When a controlling process calls *exit*(), then a SIGHUP signal is sent to each process in the foreground process group of the controlling terminal belonging to the controlling process. The remaining child processes, which are not terminated upon delivery of the SIGHUP signal, are assigned a new parent process ID. (See GA47 in 3.2.)

16(PCTS_GTI_DEVICE?A:UNTESTED)

If the function *atexit*() is supported:

When the calling process, which is a controlling process, returns from the functions previously registered by *atexit*(), then the controlling terminal associated with the session is disassociated from the session. (See GA48 in 3.2.)

Otherwise:

When the calling process calls *exit*() from a controlling process, then the controlling terminal associated with the session is disassociated from the session. (See GA48 in 3.2.)

17(C) If the function *atexit*() is supported and the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process returning from the functions previously registered by *atexit*() causes a process group to be orphaned, and when any member of the newly orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal is sent to each process in the newly orphaned process group. (See GA49 in 3.2.)

18(C) If the function *atexit*() is not supported and the behavior associated with {_POSIX_JOB_CONTROL} is supported:

When the process calling *exit*() causes a process group to be orphaned, and when any member of the newly orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal is sent to each process in the newly orphaned process group. (See GA49 in 3.2.)

8.1.50.3 Returns (8.1)

19(A) A call to *exit*() does not return.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.50.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.51 *getenv*() (8.1)

8.1.51.1 Synopsis (8.1)

The assertions for this element are identical with the assertions as specified in 4.6.1 through 4.6.4. Therefore, these assertions are all referenced assertions.

R01-R08(See Assertions 1–8 in 4.6.1 through 4.6.4.)

There are no assertions specific to this subclause.

8.1.52 *bsearch*(), *qsort*() (8.1)

8.1.52.1 Synopsis (8.1)

01(A) For *bsearch*():

If the implementation provides C Standard {2} support:

When the header <stdlib.h> is included, then the function prototype `void * bsearch(const void *, const void *, size_t, size_t, int (*) (const void *, const void * ) )` is declared. (See GA36 in 2.7.3.)

For *qsort*():

If the implementation provides C Standard {2} support:

When the header <stdlib.h> is included, then the function prototype `void qsort(void *, size_t, size_t, int (*) (const void *, const void *))` is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:

The result type for functions *bsearch*() and *qsort*() are contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(C) For *bsearch*():

If the implementation provides C Standard {2} support and *bsearch*() is defined as a macro when the header <stdlib.h> is included:

When the macro bsearch() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void* *. (See GA37 in 2.7.3.)

02(D) For *qsort*():

If the implementation provides C Standard {2} support and *qsort*() is defined as a macro when the header <stdlib.h> is included:

When the macro *qsort*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void.* (See GA37 in 2.7.3.)

See Reason 2 in Section 5. of POSIX.3 [4].

03(C) For *bsearch*():

If *bsearch*() is defined as a macro in the header <stdlib.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

For *qsort*():

If () is defined as a macro in the header <qsort>:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 1.3.3.)

8.1.52.2 Description (8.1)

04(B) For *funct*() of *bsearch*() and *qsort*():
If the implementation provides C Standard {2} support:
Element *funct*() complies with the requirements of the C Standard {2}. (See GA63 in 8..)
Otherwise:
Element *funct*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8.)
See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.52.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.52.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.53 String Handling Functions (8.1)

8.1.53.1 Synopsis (8.1)

01(A) For *funct*() of *strcpy*(), *strcat*(), and *strtok*():
If the implementation provides C Standard {2} support:
When the header <string.h> is included, then the function prototype
`char * funct(char *, const char *)` is declared. (See GA36 in 2.7.3.)
Otherwise:
When the header <string.h> is included, then the function *funct*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

For *funct*() of *strncpy*() and *strncat*():
If the implementation provides C Standard {2} support:
When the header <string.h> is included then the function prototype
`char * funct(char *, const char *, size_t)` is declared. (See GA36 in 2.7.3.)
Otherwise:
When the header <string.h> is included, then the function *funct*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

For *strcmp*():
If the implementation provides C Standard {2} support:
When the header <string.h> is included, then the function prototype
`int strcmp(const char *, const char *)` is declared. (See GA36 in 2.7.3.)
Otherwise:
When the header <string.h> is included, then the function *strcmp*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

For *strncmp*():
If the implementation provides C Standard {2} support:
When the header <string.h> is included, then the function prototype
`int strncmp(const char *, const char *, size_t)` is declared. (See GA36 in 2.7.3.)
Otherwise:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the header <string.h> is included, then the function *strncmp*() is either declared with result type *int* or not declared in the header. (See GA36 in 2.7.3.)

For *funct*() of *strchr*() and *strrchr*():

If the implementation provides C Standard {2} support:

When the header <string.h> is included, then the function prototype `char * funct(const char *, int)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <string.h> is included, then the function *funct*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

For *funct*() of *strcspn*() and *strspn*():

If the implementation provides C Standard {2} support:

When the header <string.h> is included, then the function prototype `size_t funct(const char *, const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <string.h> is included, then the function *funct*() is declared with the result type *size_t*. (See GA36 in 2.7.3.)

For *funct*() of *strpbrk*() and *strstr*():

If the implementation provides C Standard {2} support:

When the header <string.h> is included, then the function prototype `char * funct(const char *, const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <string.h> is included, then the function *funct*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

For *strlen*():

If the implementation provides C Standard {2} support:

When the header <string.h> is included, then the function prototype `size_t strlen(const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <string.h> is included, then the function *strlen*() is declared with the result type *size_t*. (See GA36 in 2.7.3.)

02(C) For *funct*() of *strcpy*(), *strcat*(), and *strtok*():

If *funct*() is defined as a macro when the header <string.h> is included:

When the macro *funct*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

For *funct*() of *strncpy*() and *strncat*():

If *funct*() is defined as a macro when the header <string.h> is included:

When the macro *funct*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

For *strcmp*():

If *strcmp*() is defined as a macro when the header <string.h> is included:

When the macro *strcmp*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

For *strncmp*():

If *strncmp*() is defined as a macro when the header <string.h> is included:

When the macro *strncmp*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

For *funct*() of *strchr*() and *strrchr*():

If *funct*() is defined as a macro when the header <string.h> is included:

When the macro *funct*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

For *funct*() of *strcspn*() and *strspn*():

If *funct*() is defined as a macro when the header <string.h> is included:

When the macro *funct*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *size_t*. (See GA37 in 2.7.3.)

For *funct*() of *strpbrk*() and *strstr*():

If *funct*() is defined as a macro when the header <string.h> is included:

When the macro *funct*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

For *strlen*():

If *strlen*() is defined as a macro when the header <string.h> is included:

When the macro *strlen*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *size_t*. (See GA37 in 2.7.3.)

03(C) If the *strcpy*(), *strncpy*(), *strcat*(), *strncat*(), *strcmp*(), *strncmp*(), *strchr*(), *strcspn*(), *strpbrk*(), *strrchr*(), *strspn*(), *strstr*(), *strtok*(), or *strlen*() function is defined as a macro in the header <string.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.53.2 Description (8.1)

04(B) For *funct*() of *strcpy*(), *strncpy*(), *strcat*(), *strncat*(), *strcmp*(), *strncmp*(), *strchr*(), *strcspn*(), *strpbrk*(), *strrchr*(), *strspn*(), *strstr*(), *strtok*(), or *strlen*():

If the implementation provides C Standard {2} support:

Element *funct*() complies with the requirements of the C Standard {2}. (See GA63 in 8..)

Otherwise:

Element *funct*() complies with the requirements of the C Standard {2} unless amended by the PCD.1. (See GA63 in 8..)

See Reason 2 in Section 5. of POSIX.3 {4}.

8.1.53.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.53.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.54 *time*() (8.1)

8.1.54.1 Synopsis (8.1)

The assertions for this element are identical with the assertions as specified in 4.5.1 through 4.5.4. Therefore, these assertions are all referenced assertions.

R01-R06 (See Assertions 1-6 in 4.5.1 through 4.5.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

There are no assertions specific to this subclause.

8.1.55 *asctime*(), *gmtime*() (8.1)

8.1.55.1 Synopsis (8.1)

01(A) For *asctime*():

If the implementation provides C Standard {2} support:

When the header <time.h> is included, then the function prototype `char * asctime(const struct tm *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <time.h> is included, then the function *asctime*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

For *gmtime*():

If the implementation provides C Standard {2} support:

When the header <time.h> is included, then the function prototype `struct tm * gmtime(const time_t *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <time.h> is included, then the function *gmtime*() is declared with the result type *struct tm* *. (See GA36 in 2.7.3.)

02(C) For *asctime*():

If *asctime*() is defined as a macro when the header <time.h> is included:

When the macro *asctime*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

For *gmtime*():

If *gmtime*() is defined as a macro when the header <time.h> is included:

When the macro *gmtime*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *struct tm* *. (See GA37 in 2.7.3.)

03(C) If the *asctime*() or *gmtime*() function is defined as a macro in the header <time.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.55.2 Description (8.1)

04(A) For *asctime*():

A call to *asctime*(*_timeptr_*) converts the data from a structure pointed to by *_timeptr_* to a 26-character string of the form "Sun Sep 16 01:03:52 1973\n\0" and returns a pointer to the string.

For *gmtime*():

A call to *gmtime*(*_timer_*) returns a pointer to a structure of type *_tm_* containing the UTC (Co-ordinated Universal Time) conversion of *_timer_*. When *_timer_* is nonnegative, and the year is greater than or equal to 1970, then a call to *gmtime*(*_timer_*) returns a pointer to a structure of type *_tm_* containing the conversion of *_timer_* according to the expression:

```
_timer_ = tm_sec +tm_min*60 +tm_hour*3600 +
        tm_yday*86400 +(tm_year -70)*31536000 +
        ((tm_year -69)/4)*86400
```

8.1.55.3 Returns (8.1)

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.55.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.56 *ctime*() (8.1)

8.1.56.1 Synopsis (8.1)

D01(A) The details on the handling of a value of the TZ environment variable that begins with a colon are contained in 8.1.1 of the PCD.1.

D02(C) If the meanings of any characters except a leading colon (:), digits, comma (,), minus (–), plus (+), and the null character in *std* or *dst* fields are documented:
The details are contained in 8.1.1 of the PCD.1.

D03(C) If the behavior of the *offset* field of the **TZ** environment variable when containing an *hours*, *minutes*, or *seconds* field whose value falls outside the limits specified in POSIX.1 {3} is documented:
The details are contained in 8.1.1 of the PCD.1.

01(A) If the implementation provides C Standard {2} support:
When the header <time.h> is included, then the function prototype
`char * ctime(const time_t *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <time.h> is included, then the function *ctime*() is declared with the result type *char* *. (See GA36 in 2.7.3.)

02(C) If *ctime*() is defined as a macro when the header <time.h> is included:
When the macro *ctime*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *char* *. (See GA37 in 2.7.3.)

03(C) If the *ctime*() function is defined as a macro:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.56.2 Description (8.1)

04(A) When **TZ** is of the form `UCT0`, then a call to *ctime*(*timer*) returns a string identical to the string returned by an equivalent call to *asctime*(*gmtime*(*timer*)).

05(A) On a call to *ctime*(), any characters except a leading colon (:), digits, comma (,), minus (–), plus (+), and the null character can be used in the TZ environment variable to designate the standard (std) and daylight saving time-zone names.

06(A) When **TZ** is of the form `TZA`-*hrs* (where *hrs* is a decimal number in the range 0-23), then a call to *ctime*(*timer*) returns a string containing a formatted time that is *hrs* hours greater than that in the string returned by an equivalent call to *asctime*(*gmtime*(*timer*)).

07(A) When **TZ** is of the form `TZA`*hrs:mins* or `TZA+`*hrs:mins* (where *hrs* is a decimal number in the range 0-23 and rains is a decimal number in the range 0–59), then a call to *ctime*(*timer*) returns a string containing a formatted time that is hrs hours and rains minutes less than that in the string returned by an equivalent call to *asctime*(*gmtime*(*timer*)).

08(A) When *TZ* is of the form `TZA`-*hrs:mins:secs* (where *hrs* is a decimal number in the range 0–23, and *mins* and *secs* are each decimal numbers in the range 0–59), then a call to *ctime*(*timer*) returns a string containing a formatted time that is *hrs* hours, *mins* minutes, and *secs* seconds greater than that in the string returned by an equivalent call *asctime*(*gmtime*(*timer*)).

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

09(A) When **TZ** is of the form `TZA0TZB`, J*start_day*, J*end_day* (where *start_day* and *end_day* are Julian day numbers in the range 1–365), then a call to *ctime*(*timer*) returns a string containing a formatted time that differs from that returned by an equivalent call to *asctime*(*gmtime*(*timer*)) in the following manner:

— For times between 02:00:00 local time on Julian day *start_day* and 01:59:59 local time on Julian day *end_day*, the time from *ctime*(*timer*) is 1 h greater than those produced by *asctime*(*gmtime*(*timer*)). (February 29th is not counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the time fields from *ctime*() and *asctime*(*gmtime*(*timer*)) are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

10(A) When **TZ** is of the form `TZAOTZB`-*hrs*, J*start_day*, J*end_day* (where *hrs* is a decimal number in the range 0–23, and *start_day* and *end_day* are Julian day numbers in the range 1–365), then a call to *ctime*(*timer*) returns a string containing a formatted time that differs from that returned by an equivalent call to *asctime*(*gmtime*(*timer*)) in the following manner:

— For times between 02:00:00 local time on Julian day *start_day* and 01:59:59 local time on Julian day *end_day*, the time from *ctime*(*timer*) is *hrs* hours greater than that produced by *asctime*(*gmtime*(*timer*)). (February 29th is not counted as a day in the computation of *start_day* and *end_day*.)

— For other times the time fields from *crime*(*timer*) and *asctime*(*gmtime*(*timer*)) are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

11(A) When **TZ** is of the form:

`TZA0TZB`, J*start_day/hrs_1*, J*end_day/hrs_2:mins:secs*

(where *hrs_1* and *hrs_2* are decimal numbers in the range 0–23, *mins* and *secs* are decimal numbers in the range 0–59, and *start_day* and *end_day* are Julian day numbers in the range 1–365), then a call to *ctime*(*timer*) returns a string containing a formatted time that differs from that returned by an equivalent call to *asctime*(*gmtime*(*timer*)) in the following manner:

— For times between *hrs_1*:00:00 local time on Julian day *start_day* and 1 s before *hrs_2:mins:secs* local time on Julian day *end_day*, the time from *ctime*(*timer*) is 1 h greater than that produced by *asctime*(*gmtime*(*timer*)). (February 29th is not counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the time fields from *ctime*(*timer*) and *asctime*(*gmtime*(*timer*)) are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12(A) When **TZ** is of the form TZA0TZB, *start_day, end_day* (where *start_day* and *end_day* are zero-based Julian day numbers in the range 0–365), then a call to *ctime*(*timer*) returns a string containing a formatted time that differs from that returned by an equivalent call to *asctime*(*gmtime*(*timer*)) in the following manner:

— For times between 02:00:00 local time on zero-based Julian day *start_day* and 01:59:59 local time on zero-based Julian day *end_day*, the time from *ctime*(*timer*) is 1 h greater than that produced by *asctime*(*gmtime*(*timer*)). (February 29th is counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the time fields from *crime*(*timer*) and *asctime*(*gmtime*(*timer*)) are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

13(A) When **TZ** is of the form:

TZA0TZB, *start_day/hrs_1, end_day/hrs_2:mins:secs*

(where *hrs_1* and *hrs_2* are decimal numbers in the range 0–23, *mins* and *secs* are decimal numbers in the range 0–59, and *start_day* and *end_day* are zero-based Julian day numbers in the range 0–365), then a call to *ctime*(*timer*) returns a string containing a formatted time that differs from that returned by an equivalent call to *asctime*(*gmtime*(*timer*)) in the following manner:

— For times between *hrs_1*:00:00 local time on zero-based Julian day *start_day* and 1 s before *hrs_2:mins:secs* local time on zero-based Julian day *end_day* the time from *ctime*(*timer*) is 1 h greater than that produced by *asctime*(*gmtime*(*timer*)). (February 29th is counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the time fields from *ctime*(*timer*) and *asctime*(*gmtime*(*timer*)) are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

14(A) When **TZ** is of the form TZA0TZB, *Mm1.n1.d1,Mm2.n2.d2* (where *m1* and *m2* are month numbers in the range 1–12, *n1* and *n2* are week numbers in the range 1–5, and *d1* and *d2* are day numbers in the range 0–6), then a call to *ctime*(*timer*) returns a string containing a formatted time that differs from that returned by an equivalent call to *asctime*(*gmtime*(*timer*)) in the following manner:

— For times between 02:00:00 local time on day *d1* of week *n1* of month *m1* and 01:59:59 local time on day *d2* of week *n2* of month *m2*, the time from *ctime*(*timer*) is 1 h greater than that produced by *asctime*(*gmtime*(*timer*)).

— For other times, the time from *ctime*(*timer*) and *asctime*(*gmtime*(*timer*)) are the same.

15(A) A call to *crime*() sets the external variable *tzname*[0] to point to the standard time-zone designation and *tzname*[1] to point to the summer time-zone designation corresponding to the current value of the **TZ** environment variable.

16(A) When the value of the **TZ** environment variable has been changed since the previous call to *ctime*(), then a call to *ctime*() uses the new value.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.1.56.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.56.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.57 *localtime*() (8.1)

8.1.57.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <time.h> is included, then the function prototype `struct tm * localtime(const time_t *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <time.h> is included, then the function *localtime*() is declared with the result type *struct tm* *. (See GA36 in 2.7.3.)

02(C) If *localtime*() is defined as a macro when the header <time.h> is included:

When the macro *localtime*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *struct tm* *. (See GA37 in 2.7.3.)

03(C) If *localtime*() is defined as a macro in the header <time.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.57.2 Description (8.1)

04(A) When **TZ** is of the form `UCT0`, then a call to *localtime*() returns a pointer to a *struct tm* in which the time fields are the same as those in the *struct tm* returned from an equivalent call to *gmtime*().

05(A) On a call to *localtime*(), any characters except a leading colon (:), digits, comma (,), minus (−), plus (+), and the null character can be used in the **TZ** environment variable to designate the standard (std) and daylight saving time-zone names.

06(A) When **TZ** is of the form `TZA`-*hrs* (where *hrs* is a decimal number in the range 0–23), then a call to *localtime*() returns a pointer to a *struct tm* in which the time fields are *hrs* hours greater than those in the *struct tm* returned from an equivalent call to *gmtime*().

07(A) When **TZ** is of the for `TZA`*hrs:mins* or `TZA+`*hrs:mins* (where *hrs* is a decimal number in the range 0–23 and *mins* is a decimal number in the range 0–59), then a call to *localtime*() returns a pointer to a *struct tm* in which the time fields are *hrs* hours and *mins* minutes less than those in the *struct tm* returned from an equivalent call to *gmtime*().

08(A) When **TZ** is of the form `TZA`-*hrs:mins:secs* (where *hrs* is a decimal number in the range 0–23, and *mins* and *secs* are each decimal numbers in the range 0–59), then a call to *localtime*() returns a pointer to a *struct tm* in which the time fields are *hrs* hours, *mins* minutes, and *secs* seconds greater than those in the *struct tm* returned from an equivalent call to *gmtime*().

09(A) When **TZ** is of the form `TZAOTZB`, J*start_day*, J*end_day* (where *start_day* and *end_day* are Julian day numbers in the range 1–365), then a call to *localtime*() returns a pointer to a *struct tm* in which the time fields differ from those returned from an equivalent call to *gmtime*() in the following manner:

— For times between 02:00:00 local time on Julian day *start_day* and 01:59:59 local time on Julian day *end_day*, the time fields from *localtime*() are 1 h greater than those produced by *gmtime*()). (February 29th is not counted as a day in the computation of *start_day* and *end_day*.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

— For other times, the time fields from *localtime*() and *gmtime*() are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

10(A) When **TZ** is of the form TZAOTZB-*hrs*, J*start_day*, J*end_day* (where *hrs* is a decimal number in the range 0–23, and *start_day* and *end_day* are Julian day numbers in the range 1—365), then a call to *localtime*() returns a pointer to a *struct tm* in which the time fields differ from those returned from an equivalent call to *gmtime*() in the following manner:

— For times between 02:00:00 local time on Julian day *start_day* and 01:59:59 local time on Julian day *end_day*, the time fields from *localtime*() are *hrs* hours greater than those produced by *gmtime*(). (February 29th is not counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the time fields from *localtime*() and *gmtime*() are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

11(A) When **TZ** is of the form:

TZAOTZB, J*start_day/hrs_1*, J*end_day/hrs_2:mins:secs*

(where *hrs_1* and *hrs_2* are decimal numbers in the range 0–2, *mins* and *secs* are decimal numbers in the range 0–59, and *start_day* and *end_day* are Julian day numbers in the range 1–365), then a call to *localtime*() returns a pointer to a *struct tm* in which the time fields differ from those returned from an equivalent call to *gmtime*() in the following manner:

— For times between *hrs_1*:00:00 local time on Julian day *start_day* and 1 s before *hrs_2:mins:secs* local time on Julian day *end_day*, the time fields from *localtime*() are 1 h greater than those produced by *gmtime*(). (February 29th is not counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the time fields from *localtime*() and *gmtime*() are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

12(A) When **TZ** is of the form TZAOTZB, *start_day,end_day* (where *start_day* and *end_day* are zero-based Julian day numbers in the range 0–365), then a call to *localtime*() returns a pointer to a *struct tm* in which the time fields differ from those returned from an equivalent call to *gmtime*() in the following manner:

— For times between 02:00:00 local time on zero-based Julian day start_day and 01:59:59 local time on zero-based Julian day *end_day*, the time fields from *localtime*() are 1 h greater than those produced by *gmtime*(). (February 29th is counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the time fields from *localtime*() and *gmtime*() are the same.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

13(A) When **TZ** is of the form:

TZAOTZB, *start_day/hrs_1, end_day/hrs_2:mins:secs*

(where *hrs_1* and *hrs_2* are decimal numbers in the range 0–23, *mins* and *secs* are decimal numbers in the range 0–59, and *start_day* and *end_day* are zero-based Julian day numbers in the range 0–365), then a call to *localtime*() returns a pointer to a *struct tm* in which the time fields differ from those returned from an equivalent call to *gmtime*() in the following manner:

— For times between *hrs_1*:00:00 local time on zero-based Julian day *start_day* and 1 s before *hrs_2:mins:secs* local time on zero-based Julian day *end_day*, the time fields from *localtime*() are 1 h greater than those produced by *gmtime*(). (February 29th is counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the time fields from *localtime*() and *gmtime*() are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

14(A) When **TZ** is of the form TZAOTZB, M*m1.n1.d1*,M*m2.n2.d2* (where *m1* and *m2* are month numbers in the range 1–12, *n1* and *n2* are week numbers in the range 1–5, and *d1* and *d2* are day numbers in the range 0–6), then a call to *localtime*() returns a pointer to a *struct tm* in which the time fields differ from those returned from an equivalent call to *gmtime*() in the following manner:

— For times between 02:00:00 local time on day *d1* of week *n1* of month *m1* and 01:59:59 local time on day *d2* of week *n2* of month *m2*, the time fields from *localtime*() are 1 h greater than those produced by *gmtime*().

— For other times, the time fields from *localtime*() and *gmtime*() are the same.

15(A) A call to *localtime*() sets the external variable *tzname*[0] to point to the standard time-zone designation and *tzname*[1] to point to the summer time-zone designation corresponding to the current value of the **TZ** environment variable.

16(A) When the value of the **TZ** environment variable has been changed since the previous call to *localtime*(), then a call to *localtime*() uses the new value.

8.1.57.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.57.4 Errors (8.1)

There are no assertions specific to this subclause.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.58 *mktime*() (8.1)

8.1.58.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header <time.h> is included, then the function prototype `time_t mktime(struct tm *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <time.h> is included, then the function *mktime*() is declared with the result type *time_t*. (See GA36 in 2.7.3.)

02(C) If *mktime*() is defined as a macro when the header <time.h> is included:

When the macro *mktime*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *time_t*. (See GA37 in 2.7.3.)

03(C) If *mktime*() is defined as a macro in the header <time.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.58.2 Description (8.1)

04(A) When **TZ**=UCT0, *_timer_* is nonnegative, and the year is greater than or equal to 1970, then a call to *mktime*() returns the value *_timer_* given by the expression:

```
_timer_ = tm_sec +tm_min*60 +tm_hour*3600 +
        tm_yday*86400 +(tm_year -70) *31536000 +
        ((tm_year -69)/4) *86400
```

where *tm_sec, tm_min, tm_hour, tm_yday* and *tm_year* are the adjusted values in the *struct tm* after the *mktime*() call has returned.

05(A) On a call to *mktime*(), any characters except a leading colon (:), digits, comma (,), minus (−), plus (+), and the null character can be used in the **TZ** environment variable to designate the standard (std) and daylight saving time-zone names.

06(A) When **TZ** is of the form TZA-*hrs* (where *hrs* is a decimal number in the range 0–23), then a call to *mktime*() returns a calendar time that is *hrs* hours less than that returned from an equivalent call to *mktime*() when **TZ** is of the form UCT0.

Testing Requirements:

Test for instances of *struct tm* with the *tm_isdst* field set to 1, 0, and -1.

07(A) When **TZ** is of the form TZA+*hrs:mins* or TZA+*hrs:mins* (where *hrs* is a decimal number in the range 0–23 and *mins* is decimal number in the range 0–59), then a call to *mktime*() returns a calendar time that is *hrs* hours and *mins* minutes greater than that returned from an equivalent call to *mktime*() when **TZ** is of the form UCT0.

Testing Requirements:

Test for instances of *struct tm* with the *tm_isdst* field set to 1, 0, and −1.

08(A) When **TZ** is of the form TZA-*hrs:mins:secs* (where *hrs* is a decimal number in the range 0–23, and *mins* and *secs* are each decimal numbers in the range 0–59), then a call to *mktime*() returns a calendar time that is *hrs* hours, *mins* minutes, and *secs* seconds less than that returned from an equivalent call to *mktime*() when **TZ** is of the form UCT0.

Testing Requirements:

Test for instances of *struct tm* with the *tm_isdst field* set to 1, 0, and −1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

09(A) When **TZ** is of the form TZAOTZB, J*start_day*, J*end_day* (where *start_day* and *end_day* are Julian day numbers in the range 1–365), and when the *tm_isdst* member of the *struct tm* is set to −1, then a call to *mktime*() returns a calendar time that differs from that returned from an equivalent call to *mktime*() when **TZ** is of the form UCT0 in the following manner:

— For times between 02:00:00 local time on Julian day *start_day* and 01:59:59 local time on Julian day *end_day*, the calendar time from a call to *mktime*() is 1 h less than that produced from an equivalent call to *mktime*() when **TZ** is of the form UCT0. (February 29th is not counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the calendar times from the two calls are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

The ambiguous times between 02:00:00 and 02:59:59 on Julian day *start_day* and 01:00:00 and 01:59:59 on Julian day *end_day* are to be tested. Results that treat these input dates either as standard time or as daylight saving time are both to be accepted.

10(A) When **TZ** is of the form TZAOTZB-hrs, J*start_day*, J*end_day* (where *hrs* is a decimal number in the range 0–23, and *start_day* and *end_day* are Julian day numbers in the range 1–365), and when the *tm_isdst* member of the *struct tm* is set to -1, then a call to *mktime*() returns a calendar time that differs from that returned from an equivalent call to *mktime*() when **TZ** is of the form UCT0 in the following manner:

— For times between 02:00:00 local time on Julian day *start_day* and 01:59:59 local time on Julian day *end_day*, the calendar time from a call to *mktime*() is *hrs* hours less than that produced from an equivalent call to *mktime*() when **TZ** is of the form UCT0. (February 29th is not counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the calendar times from the two calls are the same.

Testing Requirements:

For identification testing: Test for dates on a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

The ambiguous times between 02:00:00 and 02:59:59 on Julian day *start_day* and 01:00:00 and 01:59:59 on Julian day *end_day* are to be tested. Results that treat these input dates either as standard time or as daylight saving time are both to be accepted.

11(A) When **TZ** is of the form:

TZAOTZB, J*start_day/hrs_1*, J*end_day/hrs_2:mins:secs*

(where *hrs_1* and *hrs_2* are decimal numbers in the range 0–23, *mins* and *secs* are decimal numbers in the range 0–59, and *start_day* and *end_day* are Julian day numbers in the range 1–365), and when the *tm_isdst* member of the *struct tm* is set to −1, then a call to *mktime*() returns a calendar time that differs from that returned from an equivalent call to *mktime*() when **TZ** is of the form UCT0 in the following manner:

— For times between *hrs_1*:00:00 local time on Julian day *start_day* and 1 s before *hrs_2:mins:secs* local time on Julian day *end_day*, the calendar time from a call to *mktime*() is 1 h less than that produced from an equivalent call to *mktime*() when **TZ** is of the form UCT0. (February 29th is not counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the calendar times from the two calls are the same.

Testing Requirements:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

The ambiguous times between 02:00:00 and 02:59:59 on Julian day *start_day* and 01:00:00 and 01:59:59 on Julian day *end_day* are to be tested. Results that treat these input dates either as standard time or as daylight saving time are both to be accepted.

12(A) When **TZ** is of the form TZAOTZB, *start_day,end_day* (where *start_day* and *end_day* are zero-based Julian day numbers in the range 0–365), and the *tm_isdst* member of the *struct tm* is set to −1, then a call to *mktime*() returns a calendar time that differs from that returned from an equivalent call to *mktime*() when **TZ** is of the form UCT0 in the following manner:

— For times between 02:00:00 local time on zero-based Julian day *start_day* and 01:59:59 local time on zero-based Julian day *end_day*, the calendar time from a call to *mktime*() is 1 h less than that produced from an equivalent call to *mktime*() when **TZ** is of the form UCT0. (February 29th is counted as a day in the computation of *start_day* and *end_day*.)

— For other times, the calendar times from the two calls are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

The ambiguous times between 02:00:00 and 02:59:59 on Julian day *start_day* and 01:00:00 and 01:59:59 on Julian day *end_day* are to be tested. Results that treat these input dates either as standard time or as daylight saving time are both to be accepted.

13(A) When **TZ** is of the form:

TZAOTZB, *start_day/hrs_1, end_day/hrs_2:mins:secs*

(where *hrs_1* and *hrs_2* are decimal numbers in the range 0–23, *mins* and secs are decimal numbers in the range 0–59, and start_day and end_day are zero-based Julian day numbers in the range 0–365), and when the *tm_isdst* member of the *struct tm* is set to -1, then a call to *mktime*() returns a calendar time that differs from that returned from an equivalent all to *mktime*() when **TZ** is of the form UCT0 in the following manner:

— For times between *hrs_1*:00:00 local time on zero-based Julian day *start_day* and 1 s before *hrs_2.mins:secs* local time on zero-based Julian day *end_day*, the calendar time from a call to *mktime*() is 1 h less than that produced from an equivalent call to *mktime*() when **TZ** is of the form UCT0. (February 29th is counted as day in the computation of *start_day* and *end_day*).

— For other times, the calendar times from the two calls are the some.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

The ambiguous times between 02:00:00 and 02:59:59 on Julian day *start_day* and 01:00:00 and 01:59:59 on Julian day *end_day* are to be tested. Results that treat these input dates either as standard time or as daylight saving time are both to be accepted.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

14(A) When **TZ** is of the form `TZAOTZB,M`*m1.n1.d1*`,M`*m2.n2.d2* (where *m1* and *m2* are month numbers in the range 1–12, *n1* and *n2* are week numbers in the range 1–5, and *d1* and *d2* are day numbers in the range 0–6), and when the *tm_isdst* member of the *struct tm* is set to −1, then a call to *mktime*() returns a calendar time that differs from that returned from an equivalent call to *mktime*() when **TZ** is of the form `UCT0` in the following manner:

— For times between 02:00:00 local time on day *d1* of week *n1* of month *m1* and 01:59:59 local time on day *d2* of week *n2* of month *m2*, the calendar time from a call to *mktime*() is 1 h less than that produced from an equivalent call to *mktime*() when **TZ** is of the form `UCT0`.

— For other times, the calendar times from the two calls are the same.

Testing Requirements:

For identification testing: Test for dates in a leap year, with one or both of the start and end dates for daylight saving time occurring after 29th February, and for dates not in a leap year.

For thorough testing, changeover dates on and around 29th February must be tested: Test for dates in a leap year, with start or end dates for daylight saving time occurring on 28th February, 29th February, and 1st March; and for dates not in a leap year, with start or end dates for daylight saving time occurring on 28th February and 1st March.

The ambiguous times between 02:00:00 and 02:59:59 on Julian day *start_day* and 01:00:00 and 01:59:59 on Julian day *end_day* are to be tested. Results that treat these input dates either as standard time or as daylight saving time are both to be accepted.

15(A) A call to *mktime*() sets the external variable *tzname*[0] to point to the standard time-zone designation and *tzname*[1] to point to the summer time-zone designation corresponding to the current value of the **TZ** environment variable.

16(A) When the value of the **TZ** environment variable has been changed since the previous call to *mktime*(), then a call to *mktime*() uses the new value.

8.1.58.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.58.4 Errors (8.1)

There are no assertions specific to this subclause.

8.1.59 *strftime*() (8.1)

8.1.59.1 Synopsis (8.1)

01(A) If the implementation provides C Standard {2} support:

When the header `<time.h>` is included, then the function prototype `size_t strftime(char *, size_t, const char *, const struct tm *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header `<time.h>` is included, then the function *strftime*() is declared with the result type *size_t*. (See GA36 in 2.7.3.)

02(C) If *strftime*() is defined as a macro when the header `<time.h>` is included:

When the macro *strftime*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *size_t*. (See GA37 in 2.7.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

03(C) If *strftime*() is defined as a macro in the header <time.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.1.59.2 Description (8.1)

04(A) On a call to *strftime*(), any characters except a leading colon (:), digits, comma (,), minus (−), plus (+), and the null character can be used in the TZ environment variable to designate the standard (std) and daylight saving time-zone names.

05(A) When **TZ** is of the form **TZA**, and when *tm_isdst* is equal to 0, then a call to *strftime*() uses the string **TZA** to satisfy the conversion sequence %Z wherever this appears in the format string.

06(A) When **TZ** is of the form **TZA1TZB**, then a call to *strftime*() uses the string **TZA** to satisfy the conversion sequence %Z (wherever this appears in the format string) when *tm_isdst* is equal to 0 and uses the string **TZB** to satisfy the conversion sequence %Z when *tm_isdst* is greater than 0.

07(A) A call to *strftime*() sets the external variable *tzname*[0] to point to the standard time-zone designation and *tzname*[1] to point to the summer time-zone designation corresponding to the current value of the **TZ** environment variable.

08(A) When the value of the **TZ** environment variable has been changed since the previous call to *strftime*(), then a call to *strftime*() uses the new value.

8.1.59.3 Returns (8.1)

There are no assertions specific to this subclause.

8.1.59.4 Errors (8.1)

There are no assertions specific to this subclause.

8.2 C Language Input/Output Functions

8.2.1 *fileno*()

8.2.1.1 Synopsis

01(A) If the implementation provides C Standard {2} support:
When the header <stdio.h> is included, then the function prototype `int fileno(FILE *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <stdio.h> is included, then the function *fileno*() is either declared with result type int or not declared in the header. (See GA36 in 2.7.3.)

02(C) If *fileno*() is defined as a macro when the header <stdio.h> is included:
When the macro *fileno*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *fileno*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860438 707 ■

8.2.1.2 Description

04(A) At entry to *main*(), *stdin* can be read, and *stdout* and *stderr* can be written.

05(A) At entry to *main*(), the file position indicators associated with *stdout* and *stderr* are set to the positions indicated by the file offsets associated with file descriptors 0, 1, and 2, respectively.

06(A) At entry to *main*(), the error and end-of-file indicators associated with *stdin*, *stdout*, and *stderr* are clear.

07(A) When a call to *fileno*(*stream*) completes successfully, then the integer file descriptor associated with the argument *stream* is returned.

08(A) When the header <unistd.h> is included, then the symbols and corresponding values are as defined in Table 8.1.

Table 8.1—Standard File Descriptors

Symbol	Value
STDIN_FILENO	0
STDOUT_FILENO	1
STDERR_FILENO	2

09(A) When an application is started, then the file descriptors STDIN_FILENO, STDOUT_FILENO, and STDERR_FILENO are associated with the C Standard {2} stdin, stdout, and stderr, respectively.

8.2.1.3 Returns

10(B) When a call to *fileno*() completes unsuccessfully, then a value of (*int*)-1 is returned.

See Reason 2 in Section 5. of POSIX.3 {4}.

8.2.1.4 Errors

D01(C) If error conditions are detected for *fileno*(), and this is documented:

The details on the error conditions detected are contained in 8.2.1.4 of the PCD.1. (See DGA02 in 2.4.)

8.2.2 *fdopen*()

8.2.2.1 Synopsis

01(A) If the implementation provides C Standard {2} support:

When the header <stdio.h> is included, then the function prototype
`FILE * fdopen(int, const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <stdio.h> is included, then the function *fdopen*() is declared with the result type *FILE* *. (See GA36 in 2.7.3.)

02(C) If *fdopen*() is defined as a macro when the header <stdio.h> is included:

When the macro *fdopen*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *FILE* *. (See GA37 in 2.7.3.)

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

03(C) If *fdopen*() is defined as a macro in the header <stdio.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.2.2.2 Description

04(A) When the *fildes* argument is open for reading, then the stream returned by a call to *fdopen*(*fildes*, "*r*") can be read and not written.

05(A) When the *fildes* argument is open for writing, then the stream returned by a call to *fdopen*(*fildes*, "*w*")can be written and cannot be read, and the file length is unchanged.

06(A) When the *fildes* argument is open for writing, then the stream returned by a call to *fdopen*(*fildes*, "*a*") can be written and cannot be read, and all writes to the stream are at the end-of-file.

07(A) When the *fildes* argument is open for reading and writing, then the stream returned by a call to *fdopen*(*fildes*, "*r+*") can be both read and written.

08(A) When the *fildes* argument is open for reading and writing, then the stream returned by a call to *fdopen*(*fildes*, "*w+*") can be both written and read, and the file length is unchanged.

09(A) When the *fildes* argument is open for reading and writing, then the stream returned by a call to *fdopen*(*fildes*, "*a+*") can be both written and read, and all writes to the stream are at the end-of-file.

D01(C) If the implementation supports additional values for the *fdopen*()*type* argument, and this is documented:
The details are contained in 8.2.2.2 of the PCD.1.

10(A) The file position indicator associated with the stream returned by a call to *fdopen*() is set to the position indicated by the file offset associated with the file descriptor.

11(A) The error and end-of-file indicators associated with the stream returned by a call to *fdopen*() are cleared.

8.2.2.3 Returns

12(A) When a call to *fdopen*() completes successfully, then a pointer to a stream is returned.

13(B) When a call to *fdopen*() completes unsuccessfully, then a **NULL**. pointer is returned and sets *errno* to indicate the error.

See Reason 2 in Section 5. of POSIX.3 {4}.

8.2.2.4 Errors

D02(C) If error conditions are detected for *fdopen*(), and this is documented:
The details on the error conditions detected are contained in 8.2.2.4 of the PCD.1. (See DGA02 in 2.4.)

8.2.3 Interactions of Other *FILE*-Type C Functions

D01(C) If the results of function calls involving two or more handles that are not coordinated as described in Section 8.2.3 of POSIX.1 {3} are documented:
The details are contained in 8.2.3 of the PCD.1.

D02(C) If the state of the open file description when the active handle ceases to be accessible before the requirements on the first handle have been met are documented:
The details are contained in 8.2.3 of the PCD.1.

D03(C) If the implementation documents the results when the rules for file handles in Section 8.2.3 of POSIX.1 {3} are not followed:
The details are in 8.2.3 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

D04(A) The conditions under which applications that obey all of the rules specified in Section 8.2.3 of POSIX.1 {3} will see all input exactly once are contained in 8.2.3 of the PCD.1.

D05(C) If the results of a call to *ftell*() when the stream is opened in append mode or when the O_APPEND flag is set as a consequence of dealing with other handles on the file is documented:
The details are contained in 8.2.3.10 of the PCD.1.

8.3 Other C Language Functions

8.3.1 Nonlocal Jumps

8.3.1.1 *sigsetjmp*() (8.3.1)

8.3.1.1.1 Synopsis (8.3.1.1)

01(C) If *sigsetjmp*() is not defined as a macro:
When the header <setjmp.h> is included, then *sigsetjmp*() is declared as an identifier with external linkage and result type *int*. (See GA36 in 2.7.3.)

02(C) If *sigsetjmp* () is defined as a macro in the header <setjmp.>:
When the header <setjmp.h> is included and the macro *sigsetjmp*() is invoked with the correct argument types, then it expands to an expression with the result type *int*. (See GA37 in 2.7.3.)

03(C) If *sigsetjmp*() is defined as a macro in the header <setjmp.h>:
It protects its return value with extra parentheses when necessary. (See GA01 in 1.3.4.)

8.3.1.1.2 Description (8.3.1.2)

04(A) A call to *sigsetjmp*(*env, 0*) saves the calling environment in *env*.

05(A) When the value of the *savemask* argument is not zero, then a call to *sigsetjmp*(*env, savemask*) will save the current signal mask of the process in *env*.

06(A) A return from a direct invocation of *sigsetjmp*() returns a zero value. A return from *sigsetjmp*() caused by an invocation of *siglongjmp*() returns a nonzero value.

8.3.1.2 *siglongjmp*() (8.3.1)

8.3.1.2.1 Synopsis (8.3.1.1)

01(C) If the implementation provides C Standard {2} support:
When the header <setjmp.h> is included, then the function proto-type
`void siglongjmp(sigjmp_buf, int)` is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *siglongjmp*() is contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(D) If the implementation provides C Standard {2} support and *siglongjmp*() is defined as a macro when the header <setjmp.h> is included:
When the macro *siglongjmp*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)

See Reason 2 in Section 5. of POSIX.3 {4}.

03(C) If *siglongjmp*() is defined as a macro in the header <setjmp.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary. (See GA01 in 1.3.3.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.3.1.2.2 Description (8.3.1.2)

04(A) When the *env* argument was initialized by a call to *sigsetjmp(env, savemask)* with a nonzero *savemask* argument, then the *siglongjmp(env, val)* function restores the saved signal mask and causes the corresponding *sigsetjmp*() call to return with the value *val*, if *val* is not zero.

05(A) When *val* is zero, a call to *siglongjmp(env, val)* causes the corresponding *sigsetjmp*() to return with the value one.

06(A) When the *env* argument was initialized by a call to *sigsetjmp(env, savemask)*, then a subsequent *siglongjmp(env, val)* call restores the calling environment from *env*.

07(A) When the *env* argument was initialized by a call to *sigsetjmp(env, 0)*, then the *siglongjmp*() function leaves the signal mask of the process unchanged.

8.3.2 *tzset*()

8.3.2.1 Synopsis

01(C) If the implementation provides C Standard {2} support:
When the header `<time.h>` is included, then the function prototype `void tzset(void)` is declared. (See GA36 in 2.7.3.)

D01(C) If the implementation provides Common-Usage C support:
The result type for function *tzset*() is contained in 8.1 of the PCD.1. (See DGA01 in 1.3.3.3.)

02(C) If the implementation provides C Standard {2} support and *tzset*() is defined as a macro when the header `<time.h>` is included:
When the macro *tzset*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *void*. (See GA37 in 2.7.3.)

8.3.2.2 Description

D02(A) When **TZ** is absent, the default time-zone information used by *tzset*() is contained in 8.3.2.2 of the PCD.1.

03(A) When the header `<time.h>` is included, then the external variable tzname is defined as *extern char *tzname[2]*.

04(A) The *tzset*() function sets the external variable *tzname[0]* to point to the standard time-zone designation and *tzname[1]* to point to the summer time-zone designation corresponding to the current value of the **TZ** environment variable.

05(A) On a call to *tzset*(), any characters except a leading colon (:), digits, comma (,), minus (−), plus (+), and the null character can be used in the **TZ** environment variable to designate the standard (std) and daylight saving (dst) time-zone names.

9. System Databases

9.1 System Databases

D01(A) The interpretation of a null initial working directory field in the user database is contained in 9.1 of the PCD.1.

D02(A) The system default value to use when there is a null initial user program field specified in the user database is described in 9.1 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

STD.ISO 14515-1-ENGL 2000 ■ 4851903 0860442 138 ■

9.2 Database Access

9.2.1 Group Database Access

9.2.1.1 *getgrgid*() (9.2.1)

9.2.1.1.1 Synopsis (9.2.1.1)

01(A) If the implementation provides C Standard {2} support:

When the header <grp.h> is included, then the function prototype `struct group * getgrgid(gid_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <grp.h> is included, then the function *getgrgid*() is declared with the result type *struct group* *. (See GA36 in 2.7.3.)

02(C) If *getgrgid*() is defined as a macro when the header <grp.h> is included:

When the macro *getgrgid*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *struct group* *. (See GA37 in 2.7.3.)

03(C) If *getgrgid*() is defined as a macro in the header <grp.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

9.2.1.1.2 Description (9.2.1.2)

04(A) A call to *getgrgid(gid)* returns a pointer to a structure containing an entry from the group database whose *gr_gid* matches the argument *gid.*

05(A) When the header is included, then the elements are defined in *struct group* with the corresponding types as specified in Table 9.1.

Table 9.1—*group* Structure

Element	Type
gr_name	char *
gr_gid	gid_t
gr_mem	char **

9.2.1.1.3 Returns (9.2.1.3)

06(A) When the requested entry is not found, then the call to *getgrgid*() returns a **NULL** pointer.

07(D) If the implementation supports error conditions for *getgrgid*():

When a call to *getgrgid*() completes unsuccessfully, then a **NULL** pointer is returned to indicate the error.

See Reason 2 in Section 5. of POSIX.3 {4}.

D01(C) If the pointer returned by *getgrgid*() points to data that may be overwritten by another call to *getgrgid*(), and this is documented:

The details are contained in 9.2.1.3 of the PCD.1.

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.1.1.4 Errors (9.2.1.4)

D02(C) If error conditions are detected for *getgrgid*(), and this is documented:
The details on the error conditions detected are contained in 9.2.1.4 of the PCD.1. (See DGA01 in 2.4.)

9.2.1.2 *getgrnam*() (9.2.1)

9.2.1.2.1 Synopsis (9.2.1.1)

01(A) If the implementation provides C Standard {2} support:
When the header <grp.h> is included, then the function prototype `struct group * getgrnam(const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <grp.h> is included, then the function *getgrnam*() is declared with the result type *struct group* *. (See GA36 in 2.7.3.)

02(C) If *getgrnam*() is defined as a macro when the header <grp.h> is included:
When the macro *getgrnam*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *struct group* *. (See GA37 in 2.7.3.)

03(C) If *getgrnam*() is defined as a macro in the header <grp.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

9.2.1.2.2 Description (9.2.1.2)

04(A) A call to *getgrnam(name)* returns a pointer to a structure containing an entry from the group database whose *gr_name* matches the argument *name*.

9.2.1.2.3 Returns (9.2.1.3)

05(A) When the requested entry is not found, then the call to *getgrnam*() returns a **NULL** pointer.

06(D) If the implementation supports error conditions for *getgrnam*():
When a call to *getgrnam*() completes unsuccessfully, then a **NULL** pointer is returned to indicate the error.

See Reason 2 in Section 5. of POSIX.3 {4}.

D01(C) If the pointer returned by *getgrnam*() points to data that may be overwritten by another call to *getgrnam*(), and this is documented:
The details are contained in 9.2.1.3 of the PCD.1.

9.2.1.2.4 Errors (9.2.1.4)

D02(C) If error conditions are detected for *getgrnam*(), and this is documented:
The details on the error conditions detected are contained in 9.2.1.4 of the PCD.1. (See DGA01 in 2.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.2 User Database Access

9.2.2.1 *getpwuid*() (9.2.2)

9.2.2.1.1 Synopsis (9.2.2.1)

01(A) If the implementation provides C Standard {2} support:

When the header <pwd.h> is included, then the function prototype `struct passwd * getpwuid(uid_t)` is declared. (See GA36 in 2.7.3.)

Otherwise:

When the header <pwd.h> is included, then the function *getpwuid*() is declared with the result type *struct passwd* *. (See GA36 in 2.7.3.)

02(C) If *getpwuid*() is defined as a macro when the header <pwd.h> is included:

When the macro *getpwuid*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *struct passwd* *. (See GA37 in 2.7.3.)

03(C) If *getpwuid*() is defined as a macro in the header <pwd.h>:

It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

9.2.2.1.2 Description (9.2.2.2)

04(A) A call to *getpwuid(uid)* returns a pointer to a structure containing an entry from the user database whose *pw_uid* matches the argument *uid*.

05(A) When the header is included, then the elements are defined in *struct passwd* with the corresponding types as specified in Table 9.2.

Table 9.2—*passwd* Structure

Element	Type
pw_name	**char** *
pw_wid	**uid_t**
pw_gid	**gid_t**
pw_dir	**char** *
pw_shell	**char** *

9.2.2.1.3 Returns (9.2.2.3)

06(A) When the requested entry is not found, then the call to *getpwuid*() returns a **NULL** pointer.

07(D) If the implementation supports error conditions for *getpwuid*():

When a call to *getpwuid*() completes unsuccessfully, then a **NULL** pointer is returned to indicate the error.

See Reason 2 in Section 5. of POSIX.3 {4}.

D01(C) If the pointer returned by *getpwuid*() points to data that may be overwritten by another call to *getpwuid*(), and this is documented:

The details are contained in 9.2.2.3 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.2.1.4 Errors (9.2.2.4)

D02(C) If error conditions are detected for *getpwuid*(), and this is documented:
The details on the error conditions detected are contained in 9.2.2.4 of the PCD.1. (See DGA01 in 2.4.)

9.2.2.2 *getpwnam*() (9.2.2)

9.2.2.2.1 Synopsis (9.2.2.1)

01(A) If the implementation provides C Standard {2} support:
When the header <pwd.h> is included, then the function prototype
`struct passwd * getpwnam(const char *)` is declared. (See GA36 in 2.7.3.)

Otherwise:
When the header <pwd.h> is included, then the function *getpwnam*() is declared with the result type *struct passwd* *. (See GA36 in 2.7.3.)

02(C) If *getpwnam*() is defined as a macro when the header <pwd.h> is included:
When the macro *getpwnam*() is invoked with the correct argument types (or compatible argument types in the case that C Standard {2} support is provided), then it expands to an expression with the result type *struct passwd* *. (See GA37 in 2.7.3.)

03(C) If *getpwnam*() is defined as a macro in the header <pwd.h>:
It evaluates its arguments only once, fully protected by parentheses when necessary, and protects its result value with extra parentheses when necessary. (See GA01 in 1.3.4.)

9.2.2.2.2 Description (9.2.2.2)

04(A) A call to *getpwnam(name)* returns a pointer to a structure containing an entry from the user database whose *pw_name* matches the argument *name*.

9.2.2.2.3 Returns (9.2.2.3)

05(A) When the requested entry is not found, then the call to *getpwnam*() returns a **NULL** pointer.

06(D) If the implementation supports error conditions for *getpwnam*():
When a call to *getpwnam*() completes unsuccessfully, then a **NULL** pointer is returned to indicate the error.

See Reason 2 in Section 5. of POSIX.3 {4}.

D01(C) If the pointer returned by *getpwnam*() points to data that may be overwritten by another call to *getpwnam*(), and this is documented:
The details are contained in 9.2.2.3 of the PCD.1.

9.2.2.2.4 Errors (9.2.2.4)

D02(C) If error conditions are detected for *getpwnam*(), and this is documented:
The details on the error conditions detected are contained in 9.2.2.4 of the PCD.1. (see DGA01 in 2.4.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

10. Data Interchange Format

10.1 Archive/Interchange File Format

D01(A) The interface to the format-creating and format-reading utilities used to translate to/from the formats defined in Section 10.1 of POSIX.1 {3}, including the names of the utilities, are contained in 10.1 of the PCD.1.

R01 A conforming system provides a mechanism to copy files from a medium to the file hierarchy using the extended `cpio` format. (See Assertion 21 in 10.1.2.5.)

01(A) A conforming system provides a mechanism to copy files from a file hierarchy to a medium using the extended `cpio` format.

02(A) A conforming system provides a mechanism to copy files from a file hierarchy to a medium using the extended `tar` format.

03(A) A conforming system provides a mechanism to copy files from a medium to the file hierarchy using the extended `tar` format.

04(B) When used by a process without the appropriate privileges, then the extended `cpio` format-reading utility sets the user ID of an extracted file to the effective user ID of the process.

See Reason 1 in Section 5. of POSIX.3 {4}.

05(B) When used by a process without the appropriate privileges, then the extended cpie format-reading utility sets the group ID of an extracted file to the effective group ID of the process or to the group owner of its parent directory.

See Reason 1 in Section 5. of POSIX.3 {4}.

06(B) When used by a process without the appropriate privileges, then the extended `cpio` format sets the file permission mode of an extracted file to the matching file permission value as supplied by the *c_mode* field of the extended `cpio` format archived file, modified appropriately by the current *umask*() value.

See Reason 1 in Section 5. of POSIX.3 {4}.

07(B) When used by a process without the appropriate privileges, then the extended `tar` format-reading utility sets the user ID of an extracted file to the effective user ID of the process.

See Reason 1 in Section 5. of POSIX.3 {4}.

08(B) When used by a process without the appropriate privileges, then the extended `tar` format-reading utility sets the group ID of an extracted file to the effective group ID of the process or to the group owner of its parent directory.

See Reason 1 in Section 5. of POSIX.3 {4}.

09(B) When used by a process without the appropriate privileges, then the extended `tar` format-reading utility sets the file permission mode of an extracted file to the matching file permission value as supplied by the *mode* field of this format, modified appropriately by the current *umask*() value.

See Reason 1 in Section 5. of POSIX.3 {4}.

10(D) If the implementation provides the mechanism for creating a process with the appropriate privileges to restore the owner and permissions exactly as recorded on the data interchange archive:

> When the archive is an extended `cpio` format archive or an extended `tar` format archive in which the *magic* field is not "ustar", and when used by a process with the appropriate privilege to restore the owner and permissions exactly as recorded on the archive, then the data interchange format utility restores the ownership and permissions exactly as recorded on the archive.

See Reason 1 in Section 5. of POSIX.3 {4}.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11(D) If the implementation provides the mechanism for creating a process with the appropriate privileges to restore the owner and permissions exactly as recorded on the extended `tar` format archive:

When the extended `tar` format archive contains a value for the magic field of "ustar", and when used by a process with the appropriate privilege to restore the owner and permissions exactly as recorded on the archive, then the extended `tar` format utility restores the permissions exactly as recorded on the archive, and restores the ownership to the values contained in the *uname* and *gname* fields if a scan of the password and group databases matches the values contained within the *uname* and *gname* fields; otherwise, it restores the ownership to the values contained in the *uid* and *gid* fields.

See Reason 1 in Section 5. of POSIX.3 [4].

D02(C) If the media format and the frames on the media in which the data appear are documented:

The details are contained in 10.1 of the PCD.1.

10.1.1 Extended `tar` Format

R01 The extended `tar` format-creating utility can create an extended `tar` format archive. (See Assertions 2-4 in 10.1.1.)

01(A) The archive created by the extended `tar` format-creating utility contains a series of fixed-size blocks of 512 B.

02(A) The archive created by the extended `tar` format utility contains, for each file archived, a header block as defined in Table 10.1 followed by zero or more blocks that provide the contents of the file.

Table 10.1—`tar` Format

Field Name	Byte Offset	Length (bytes)
name	0	100
mode	100	8
uid	108	8
gid	116	8
size	124	12
mtime	136	12
chksum	148	8
typeflag	156	1
linkname	157	100
magic	257	6
version	263	2
uname	265	32
gname	297	32
devmajor	329	8
devmajor	337	8
prefix	345	155
(unspecified)	500	12

03(A) The archive created by the extended `tar` format-creating utility contains at its end two blocks filled with binary zeros.

D01(C) If the contents of the blocks on a tape after the two zero-filled end-ofarchive indicator blocks is documented:

The details are contained in 10.1.1 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

04(B) In the archive created by the extended `tar` format-creating utility, each group of "n" blocks may be written with a single *write*() operation. On magnetic tape, the result of this write is a single tape record.

See Reason 6 in Section 5. of POSIX.3 [4].

05(A) When the header <`tar.h`> is included, then the symbolic constants and corresponding values are as defined in Table 10.2.

Table 10.2—`tar` Symbolic Constants

Symbol	Value
TMAGIC	"ustar"
TMAGLEN	6
TVERSION	"00"
TVERSLEN	2

06(A) When the header <`tar.h`> is included, then the *typeflag* field symbolic constants with corresponding values are as defined in Table 10.3.

Table 10.3—`tar` *typeflag* Field

Symbol	ISO/IEC 646 IRV Value
REGTYPE	'0'
AREGTYPE	'\0'
LNKTYPE	'1'
SYMTYPE	'2'
CHRTYPE	'3'
BLKTYPE	'4'
DIRTYPE	'5'
FIFOTYPE	'6'
CONTTYPE	'7'

07(A) When the header <`tar.h`> is included, then the *mode* field symbolic constants with corresponding values are as defined in Table 10.4.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 10.4—`tar` *mode* Field

Symbol	Octal Value
TSUID	04000
TSGID	02000
TSTVX	01000
TUREAD	00400
TUWRITE	00200
TUEXEC	00100
TGREAD	00040
TGWRITE	00020
TGEXEC	00010
TOREAD	00004
TOWRITE	00002
TOEXEC	00001

D02(C) If an implementation supports the use of characters outside the portable filename character set in names for files, users, or groups:

The character encoding used for these names is contained in 10.1.1 of the PCD.1.

08(D) If the implementation supports an extended character set beyond the portable character set:

When an extended character set beyond the portable character set is used, and when the format-reading and format-creating utilities on the two distinct systems use the same extended character set, then the file name is preserved.

See Reason 1 in Section 5. of POSIX.3 (4).

09(A) The mechanism that produces an extended `tar` archive format supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

10(A) The mechanism that produces an extended `tar` archive format retains the unique identity of the upper- and lowercase letters of the filename. (See GA03 in 2.2.2.32.)

D03(A) Whether the data from a file, which if used would create an invalid file name is stored on the file hierarchy and under what name it is stored is contained in 10.1.1 of the PCD.1.

11(D) If a file is found on the medium that would create an invalid name, and if the format-reading utility chooses to ignore the file:

The format-reading utility produces an error indicating that the file is being ignored.

See Reason 1 in Section 5. of POSIX.3 (4).

12(A) In the archive created by the extended `tar` format-creating utility, the fields *magic*, *uname*, and *gname* are null-terminated character strings.

13(A) In the archive created by the extended `tar` format-creating utility, the fields *name*, *linkname*, and *prefix* are null-terminated character strings except when all characters in the array contain non-null characters including the last character.

14(A) In the archive created by the extended `tar` format-creating utility, the *version* field is 2 B containing the characters '00'.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

15(A) In the archive created by the extended `tar` format-creating utility, the fields *mode, uid, gid, size, mtime, chksum, devmajor,* and *devminor* are leading zero-filled octal numbers as in ISO/IEC 646 :1991{1} IRV and are terminated by one or more space or null characters.

16(A) When the *prefix* field is not an empty string, then the pathname of the file extracted by the extended `tar` format-reading utility is the concatenation of the prefix, a slash character, and the *name.*

17(A) When the *prefix* field is an empty string, then the pathname of the file extracted by the extended `tar` format-reading utility is *name.*

18(A) When an archived link is extracted by the extended `tar` format-reading utility, the *linkname* field contains the pathname (the *prefix* field is not used).

19(A) When the linkname does not fit in the space provided (100 characters), then the extended `tar` format-creating utility does not store the link on the archive.

D04(C) If the implementation documents whether the format-reading utility ignores mode bits in the archive format that are not mentioned elsewhere in POSIX.1 {3}:

The documentation is in 10.1.1 of the PCD.1.

20(C) If the implementation requires appropriate privilege to set specific mode bits:

When the process restoring the files from an extended `tar` archive does not have the required appropriate privilege to set a given mode bit, and when this privilege is required, then the mode bits for which the process does not have these privileges are ignored.

21(A) In the archive created by the extended `tar` format-creating utility, the *uid* and *gid* fields are the user ID and group ID of the owner and group of the file, respectively.

22(A) In the archive generated by the extended `tar` format-creating utility, when the *typeflag* field is '0' denoting a regular file, then the *size* field is the size of the file in bytes.

23(A) In the archive generated by the extended `tar` format-creating utility, for files where *typeflag* is '1' or '2', the *size* field is zero.

24(A) In the archive generated by the extended `tar` format-creating utility, a *typeflag* field of DIRTYPE or FIFOTYPE causes no data blocks to be stored.

25(C) If the implementation supports character special files:

In the archive generated by the extended `tar` format-creating utility, a *typeflag* field of CHRTYPE causes no data blocks to be stored.

26(C) If the implementation supports block special files:

In the archive generated by the extended `tar` format-creating utility, a *typeflag* field of BLKTYPE causes no data blocks to be stored.

27(A) When a file extracted by the extended `tar` format-reading utility contains a *typeflag* field of FIFOTYPE, then the size field is ignored.

D05(C) If the *typeflag* field settings of CHARTYPE, BLKTYPE, or FIFOTYPE are documented:

The details explaining the meaning of the size field are contained in 10.1.1 of the PCD.1.

28(A) In the archive created by the extended `tar` format-creating utility, the *mtime* field is the ISO/IEC 646:1991{1} IRV representation of the octal value of the modification time of the file at the time it was archived.

29(A) In the archive created by the extended `tar` format-creating utility, the *chksum* field is the ISO/IEC 646 :1991{1} IRV representation of the octal value of the simple sum of all bytes in the header block. (Each 8 b byte in the header is treated as an unsigned value. These values are added to an unsigned integer, initialized to zero. The *chksum* field is treated as if it were all blanks.)

30(D) If the implementation, when extracting an extended `tar` format archive, supports a *typeflag* of '\0' to be a regular file:

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When reading an extended `tar` format archive, a *typeflag* of '\0' specifies a regular file.

NOTE — POSIX.1 {3} says *should*, which generates a conditional assertion. But POSIX.1 {3} provides no announcement mechanism for this condition, and the test is meaningless without a required outcome.

See Reason 2 in Section 5. of POSIX.3 {4}.

31(A) In the archive created by the extended `tar` format-creating utility, a regular file contains a *typeflag* field value of '0'.

32(A) In the archive created by the extended `tar` format-creating utility, a link to a file that is already in the archive contains a *typeflag* field value of '1'.

33(C) If the implementation supports character special files:
In the archive created by the extended `tar` format-creating utility, a character file contains a *typeflag* field value of '3'.

D06(C) If the format for the *devmajor* and *devminor* fields is documented:
The details are contained in 10.1.1 of the PCD.1.

34(C) If the implementation supports block special files:
In the archive created by the extended `tar` format-creating utility, a block special file contains a *typeflag* field value of '4'.

35(A) In the archive created by the extended `tar` format-creating utility, a directory file contains a *typeflag* field value of '5'.

36(A) In the archive created by the extended `tar` format-creating utility, a FIFO file contains a *typeflag* field value of '6'.

37(D) If the implementation supports high-performance files:
In the archive created by the extended `tar` format-creating utility, a high-performance file contains a *typeflag* field value of '7'.

See Reason 3 in Section 5. of POSIX.3 {4}.

38(A) When reading an archive, then a *typeflag* of '0' indicates to the extended `tar` format-reading utility that the archived file is a regular file.

39(A) When reading an archive, then a *typeflag* of '1' indicates to the extended `tar` format-reading utility that the archived file is a link to another file contained in the archive whose pathname is specified in the *linkname* field.

40(A) When reading an archive, then a *typeflag* of '5' indicates to the extended `tar` format-reading utility that the archived file is a directory.

41(A) When reading an archive, then a *typeflag* of '6' indicates to the extended `tar` format-reading utility that the archived file is a FIFO special file.

42(D) If the implementation supports high-performance attribute files:
When reading an archive, then a *typeflag* of '7' indicates to the extended `tar` format-reading utility that the archived file is a high-performance file.

See Reason 1 in Section 5. of POSIX.3 {4}.

43(D) If the implementation does not support high-performance files:
When reading an archive, then a *typeflag* of '7' indicates to the extended `tar` format-reading utility that the archived file is to be treated as a regular file.

See Reason 2 in Section 5. of POSIX.3 {4}.

44(C) If the implementation supports an extended `tar` format-reading utility that generates an archive in which the *magic* field is set to "ustar":

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

When the archive-creating utility generates an archive in which the *magic* field is set to "ustar", then the *uname* and *gname* fields contain the ISO/IEC 646 :1991{1} IRV representation of the owner and group of the file, respectively (truncated to fit if necessary).

10.1.1.1 Multiple Volumes (10.1.3)

45(B) When data represented by the extended `tar` format resides in more than one file, then the next logical byte in the archive is the first byte in the next file, and the extended `tar` format-reading utility is able to read such an archive correctly.

See Reason 2,4 in Section 5. of POSIX.3 {4}.

10.1.2 Extended `cpio` Format

R01 The extended `cpio` archive format is a series of entries, each comprised of a header that describes the file, the name of the file, and then the contents of the file. (See Assertion 2 in 10.1.2.1.)

01(A) Each entry in the `cpio` archive created by the extended cpio format-creating utility contains data as shown in Table 10.5.

Table 10.5—`cpio` Data

Field	Length
c_magic	6B
c_dev	6B
c_ino	6B
c_mode	6B
c_uid	6B
c_gid	6B
c_nlink	6B
c_rdev	6B
c_mtime	11B
c_namesize	6B
c_filesize	11B
c_name	c_namesize
c_filedata	c_filesize

10.1.2.1 `cpio` Header

02(A) In the archive created by the extended `cpio` format-creating utility, the header fields *c_dev, c_ino, c_mode, c_uid, c_gid, c_nlink, c_rdev, c_mtime, c_namesize,* and *c_filesize* are written as streams of ISO/IEC 646 :1991 {1} IRV characters interpreted as octal numbers. The octal numbers are extended to the necessary length by appending ISO/IEC 646 :1991{1} IRV zeros at the most-significant digit end of the number; the result is written to the stream of bytes most significant digit first.

03(A) In the archive created by the extended `cpio` format-creating utility, the *c_magic* field contains the magic bytes '070707'.

04(A) In the archive created by the extended `cpio` format-creating utility, the *c_dev* and *c_ino* pair contains values that uniquely identify the file within the archive unless they are links to the same file.

D01(C) If the method for determining the values of the *c_dev* and *c_ino* fields is documented:
The details are contained in 10.1.2.1 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

05(A) In the archive created by the extended `cpio` format-creating utility, the *c_mode* field contains values for file type and access permissions for files stored on the archive.

06(A) In the archive created by the extended `cpio` format-creating utility, the *c_uid* field contains the user ID of the owner for the file.

07(A) In the archive created by the extended `cpio` format-creating utility, the *c_gid* field contains the group ID of the group for the file.

08(A) In the archive created by the extended `cpio` format-creating utility, the *c_nlink* field contains the number of links referencing the file for the file at the time the archive was created.

D02(A) The information for character or block special files contained in *c_rdev* is contained in 10.1.2.1 of the PCD.1.

09(B) In the archive created by the extended `cpio` format-creating utility, the *c_rdev* field contains implementation-defined information for character or block special files.

See Reason 4 in Section 5. of POSIX.3 [4].

10(A) In the archive created by the extended `cpio` format-creating utility, the *c_mtime* field contains the last modification time of the file.

11(A) In the archive created by the extended `cpio` format-creating utility, the *c_namesize* field contains the length of the pathname, including the terminating null byte.

12(A) In the archive created by the extended `cpio` format-creating utility, the *c_filesize* field contains the length of the data in the file.

10.1.2.2 `cpio` File Name

13(A) In the archive created by the extended `cpio` format-creating utility, the *c_name* field follows immediately after the header and contains the pathname of the file.

D03(A) Whether the data from a file, which if used would create an invalid file name, is stored in the file hierarchy and under what name it is stored is specified in 10.1.2.2 of the PCD.1.

14(A) The mechanism that produces an extended `cpio` archive format supports *filenames* containing any of the following characters:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -

(See GA02 in 2.2.2.32.)

15(A) The mechanism that produces an extended `cpio` archive format retains the unique identity of the upper- and lowercase letters of the filename. (See GA03 in 2.2.2.32.)

16(D) If the implementation supports an extended character set beyond the portable character set:

When an extended character set beyond the portable character set is used, and when the format-reading and format-creating utilities on the two distinct systems use the same extended character set, then the file name is preserved.

See Reason 1 in Section 5. of POSIX.3 [4].

10.1.2.3 `cpio` File Data

R02 In the archive created by the extended `cpio` format-creating utility, following the *c_name* field, there are *c_filesize* bytes of data. (See Assertion 1 in 10.1.2.)

R03 In the archive created by the extended `cpio` format-creating utility, when the *c_filesize* value is zero, then no data is contained in *c_filedata*. (See Assertion 1 in 10.1.2.)

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

10.1.2.4 cpio Special Entries

17(A) In the archive created by the extended cpio format-creating utility, the value of the *c_filesize* field is zero for FIFO special fries.

18(A) In the archive created by the extended cpio format-creating utility, the value of the *c_filesize* field is zero for directories.

19(A) In the archive created by the extended cpio format-creating utility, the value of the *c_filesize* field is zero for the trailer.

D04(C) If the value of *c_filesize* for special files other than FIFO, directories, and trailer files is documented:
The details are contained in 10.1.2.4 of the PCD.1.

R04 The header for the next file entry is written directly after the last byte of the file entry preceding it. (See Assertion 1 in 10.1.2.)

20(A) In the archive created by the extended cpio format-creating utility, the header with the file name string of "TRAILER! ! !" indicates the end of the archive.

D05(C) If the contents of bytes in the last block of the archive following the header denoting the file name "TRAILER! ! !" are documented:
The details are contained in 10.1.2.4 of the PCD.1.

10.1.2.5 cpio Values

21(A) The extended cpio format-creating/reading utility can store/extract directories, FIFOs, and regular files on/from a medium, respectively.

D06(C) If file types other than C_ISDIR, C_ISFIFO, and C_ISREG that may be stored in or extracted from a cpio archive are documented:
The details are contained in 10.1.2.5 of the PCD.1.

22(D) If the implementation requires appropriate privileges to create a file of a specified type:
When the user does not have the required appropriate privilege to create a file of the specified type, then the extended cpio format-reading utility ignores the entry and issues an error to the standard error output.

See Reason 2 in Section 5. of POSIX.3 (4).

23(A) When regular files in the archive meet user selection criteria placed on the extended cpio format-reading utility, the files are restored, unless the extended cpio format-reading utility detects an error.

24(C) If the implementation requires appropriate privileges to set specific mode bits:
When the utility restoring the files from the archive does not have the required appropriate privileges to set a given mode bit, and when this privilege is required, then the mode bits for which the process does not have these privileges are ignored.

10.1.3 Multiple Volumes

25(B) When data represented by the extended cpio format resides in more than one file, the byte following the end-of-file will be the first byte on the next file, and the extended cpio format-reading utility is able to read such an archive correctly.

See Reasons 2 and 4 in Section 5. of POSIX.3 (4).

D07(A) The manner in which the format-reading utility determines what file to read as the next file following an end-of-file or end-of-media condition is described in 10.1.3 of the PCD.1.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex A Bibliography

(Informative)

A.1 Related Open Systems Standards

{B1}ISO 8348:1987 *Information processing systems—Data communications—Network service definition.*[6]

{B2}IEEE Std 100-1992 *The New IEEE Standard Dictionary of Electrical and Electronics Terms (ANSI).*[7]

[6]ISO/IEC documents can be obtained from the ISO office, 1, rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse.
[7]IEEE documents can be obtained from the The Institute of Electrical and Electronics Engineers, Inc, 345 East 47th Street, New York, New York 10017, USA.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex B Rationale and Notes

(Informative)

B.1 General

B.1.3 Conformance

NOTE — The subIcause numbers of this annex are not sequenctial because the standared developeers realized that rationale was not necessary for every subclause. If any subclause is missing, then no rationale exists for that subclause.

B.1.3.1 Implementation Conformance

B.1.3.1.1 Requirements

The reason the standard developers decided to use the terminology "If the implementation supports {_POSIX_JOB_CONTROL} ... rather than "If {_POSIX_JOB_CONTROL} is defined ..." is that the latter phrase does not convey the requirement to use the *sysconf*() and *pathconf*() functions to establish whether the facility is implemented.

Assertions for the [EINVAL] error, generated due to an invalid UID or GID, are not portably testable and therefore must be extended assertions (B and D types). UID and GID are required to be arithmetic types, but they are not required to be signed. Therefore, on implementations where they are unsigned, it is not possible to pass a negative number to a function, because it will be automatically cast to an unsigned positive integer. When implementations consider all positive integers (including zero) valid, there is no way to cause [EINVAL] to be generated on such implementations.

B.1.3.1.2 Documentation

The basis for documentation assertions is 1.3.1.2 of POSIX.1 {3}. This subclause specifies all the requirements for a conformance document. These requirements place a heavy burden on the documentation assertion writers to be complete. It is imperative that all possible data belonging in the conformance document be represented by documentation assertions. The only text allowed in a conformance document is text related to the documentation assertions.

B.1.4 Test Methods

This standard defines specific test methods to measure conformance of operating system implementations and associated conformance document specifications detailed in POSIX.1 {3}.

There has been considerable debate concerning the appropriate scope of the work represented by this standard. It was agreed upon by the standard developers to define test methods for the functionality defined in POSIX.1 {3}. The standard developers were careful to define the functionality of POSIX.1 {3} without specifying additional functionality, which would be outside the scope of this standard. The method chosen was the definition of requirements for a POSIX.1 {3} Conformance Test Suite (PCTS.1). To what extent this PCTS.1 specification would be recommended practice versus required practice came out of considerable debate. To the extent that a definition was defined as a requirement in POSIX.1 {3}, this was also translated into a requirement in this standard. Where functionality was optional, as in the use of the words may or should or implementation defined, these would be options or recommended practice, based upon the consensus of the standard developers. The manner to deinfe specific test methods for testing conformance by means of a PCTS.1 is the developement of test assertion statements derived from POSIX.1 {3} functionality (i.e., *open*(), *creat*(), ...).

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Considerable debate has taken place on the definition of a documentatoin audit as a subset of the defined test method of this standard. There have been discussions concerning the statements in POSIX.1 {3} related to the conformance document required with all operating system implementations. Since these statements are part of POSIX.1 {3}, it was assumed that a method or procedure must be documented in this standard related to these sections. Previously they were to be documented in a chapter called Documentation Audit Procedures. This was to include specifications defined in Section 1. of POSIX {3}, including specification of implementation-defined items and a manual audit procedure as part of the test method of POSIX.1 {3}. It was the opinion of the POSIX.1 standard developers that these statements should be documented in the sections where they are defined in POSIX.1 {3}. There was also considerable debate as to the nature of these documentation statements and how they should be written. The standard developers created test assertions for the testing of functions defined in POSIX.1 {3}, which will be requirements for a mechanized conformance test suite or PCTS.1. Discussions revolved around the nature of these documetation statements: are they similar to test assertion statements, since they are assertive, required statements, or should they be defined in a procedural manner as they represent a manual audit process rather than a mechanized method defined by test assertions. The final decision was to write these statements as documentation assertions.

In the case of the documentation audit and functionality requirements of a PCTS.1, this standard defines "what is to be tested for conformance" but *not* "how that is to be accomplished." The standard developers also needed to avoid areas that would be considered value-added requirements to either the PCTS.1 or the documentation audit procedure, which would be added by a vendor for differentiation and competition of product offerings to the industry.

The defined test method in this standard specifies test assertions and documentation assertions according to the functionality and requirements defined in POSIX.1 {3}.

Defining a verification method for measuring conformance to POSIX.1 {3} provides the industry with a metric for providing a compatible operating system base. Providing future measuring conformance methods for other POSIX standards would ensure compliance of new facility extensions to the existing base.

B.1.4.4 PCTS Execution

B.1.4.4.1 PCTS Target System Support Facilities

The standard developers did not examine those assertions that might become conditional assertions because of their dependence on a target system support facility. An individual assertion is not conditional just because it may require a target system support facility.

Early in the development of this standard, the standard developers recognized the problems that were inherent in testing POSIX.1 {3}. The lack of system administration functions caused considerable difficulties in generating tests for certain defined functions. The standard developers had to decide whether to mark all of these tests as extension tests or to describe a method by which outside services could be made available to a test. The latter course was chosen because it was deemed important to categorize many of these tests as base assertions rather than leaving the creators of a PCTS.1 the option of omitting these tests from the PCTS.1.

It was also accepted that many of the suites developed would be targeted at a limited set of implementations where the required facilities could be provided in a standard way.

An example of the need of a target system support facility is in a test to establish whether the error condition [EROFS] is returned when attempting an updating operation on a read-only file system. It is necessary for a PCTS.1 to control the mounting and unmounting of the read-only file system in order to undertake the test for the error return.

B.1.4.4.2 Test Tolerances

The techniques used in a PCTS.1 to measure certain functions may accept the "best that is possible" rather than apply a carefully devised assessment of the accuracy of the test. For example, in testing for the reporting of child process times, the accuracy of the values reported cannot be measured, but the fact that something is reported can be asserted.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Likewise, the fact that the reported child process times do not change until a waited-for child process has exited can also be asserted.

There are a number of areas where the intent of POSIX.1 {3} is clear, but there is no definitive statement in POSIX.1 {3} to address this. For example, it is clearly the intent of POSIX.1 {3} that the delivery of unblocked signals should be in a timely manner. A system that delayed delivery of signals for 20 s could be argued to be conforming to the letter of POSIX.1 {3}, but is clearly not conforming to the intent of POSIX.1 {3}.

B.1.4.4.3 Timeout Tolerances

The PCTS.1 needs to take account of any scheduling delays on the implementation, and clearly a degree of latitude is acceptable. The fact that an alarm signal occurs 2 s late may not be a problem in the implementation. However, if it persistently arrived 2 s late or if it arrived 5 s late, this would suggest a problem with the implementation. The PCTS.1 should be able to deal with these variances between systems and correctly assess whether or not the system is deficient.

There are several otherwise testable assertions in POSIX.1 {3} that cannot be tested reliably because a POSIX.1 {3} implementation cannot measure these time-critical features accurately. For example, POSIX.1 {3} requires *tms_utime*, (the user CPU time) to be set to zero in the child, but the user process must execute-i.e., after return from the *fork*()—and call the *times* function before the user time can be examined. Consequently, *tms_utime* will never test as zero.

There are several classes of time-critical features. One class includes the baud rate of the control modes in the general terminal interface. Measuring the baud rate can be accomplished by looping the terminal connector back to another connector on the same system and timing how long it takes a fixed-length record to be transmitted at various baud rates. In this case, one assumes that the baud rate is reflected in the data flow rate. Because of undefined latencies in the transmission process, a large latitude must be allowed for interpreting the results of such a test.

Another class of time-critical features are those that are caused by physical conditions rather than programmatic conditions. The errors [ENXIO] or [EBUSY] are examples of this. It is acceptable for a device to become momentarily unavailable, returning [ENXIO], in a nondeterministic manner. Since it is not possible to initiate this nonavailability from a program strictly using POSIX.1 {3}, such a test cannot be written. Another example is the terminal control function to flush data written but not transmitted. Since there is no function to indicate the amount of data that has been transmitted or that is still in the queue to be transmitted, there is no reliable method to verify that the flush command actually works. Since timing information is not specified in the standard, it might be improper to assume that the triggering of any sanity timeouts in a test is necessarily an indication of nonconformance.

B.1.4.6 PCTS Limits and Configuration Variables

B.1.4.6.1 Requirements

An implementation may have an extremely large limit because it uses dynamic allocation of resources to satisfy the demands of a process. This could mean that the traditional test method of grabbing resources until the limit is attained will not work or will take an exceedingly long period of time. To resolve these problems, the standard developers established the mechanism of altering the behavior of certain tests according to the limit value defined by the implementation. For example, if {OPEN_MAX} is allowed to grow to the value of {INT_MAX}, it would take an extremely long time on a 32-b based machine to prove that the correct error return was provided when the limit was exceeded. It would also be most unlikely that any application would want to open that many files, so testing for this does nothing to aid application portability. In cases where the limit was large, the assertion was modified to provide some confidence that the implementation will support a reasonably large number of files (i.e., 256).

In Section 2.9 of POSIX.1 {3}, the minimum/maximum values for system-defined limits are given. A conforming implementation may define these symbols to be greater than or equal to the value given in the standard. Many of these symbolic constants are included for consistency with ISO/IEC 9899 :1990 {2}. Some of the symbolic constants are definitions of maximum and minimum values for C-language data types. Testing conformance for these constants is simply an assignment of the respective value to the appropriate data type. Some of the constants, however, are more

 Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

difficult to test, and in some cases it may be unreasonable to attempt to do so. In some cases, the magnitude of the limit may be sufficiently large that the physical characteristics of the system will never allow that limit to be achieved. In other cases, the physical characteristics of the system may be sufficient, but the time required to achieve the limit by a test program is too great with respect to the run time for all the tests.

A conforming portable application must not require limits outside the range specified in POSIX.1 {3}. A conforming portable application may utilize any additional value if the implementation provides more than the minimum/maximum in the standard, but it cannot require values larger than those in the standard. The PCTS.1 should be designed as a conforming portable application. Therefore, it cannot require an implementation to exceed the limits defined within the standard. If the implementation provides service in excess of the standard, it would be advantageous to test to the implementation-defined limit; however, it would be incorrect to assume that the limit can always be attained. If the test suite attempts to test the system-defined limit and fails due to physical limitations, nothing has been gained in measuring conformance to the standard. For the test program to determine some reasonable limit to test, it would require extensive knowledge of the physical characteristics of the system and how efficiently they are managed (i.e., for {LINK_MAX}, the test would have to know the available disk space and how much room is needed on the disk for one link). This information would have to be provided by the tester because it would vary from system to system, and with each implementation. This information would not be easily attained, nor has the test really tested any conformance to the system-supplied limit.

To test strict conformance, the test suite should not test beyond the POSIX.1 {3} defined minimum/maximum values. By restricting the test suite to the limits in the standard, it is truly a conforming portable application for all POSIX.1 {3} implementations. It would be advantageous if the tests were written with the limits in the standard as the default, with the option to override these values by the test operator at configuration time. This method of testing allows measures of strict conformance as well as optional system-defined limits for implementation developers or potential buyers of a POSIX.1 {3} system. For system buyers, this is a valid concern because all application programs cannot be conforming portable applications and, in some cases, the buyer may have a requirement above the POSIX.1 {3} defined limit. To measure conformance, the test suite would be run without the optional values. If optional values were provided, the suite should flag any limits below the POSIX.1 {3} defined limit with a warning and report the results of the test. If optional values cannot be attained, the suite should mark the test as a failure and give the associated *errno*. If the optional values are attained, the test is marked as passing. In the case when the optional limit is too large, it is the responsibility of the tester to kill the test program and restart it with a reasonable limit.

B.1.4.7 Test Methods for Headers

B.1.4.7.1 Requirements

The requirements definition is a summary of the various assertions concerning the different elements that can appear in a header. These assertions are scattered throughout the assertion list, and the generic statements made in this section are provided to assist a PCTS.1 author in identifying the test method requirements for header file elements. The assertions were scattered throughout the assertion list because this aligned the assertion list more closely with POSIX.1 {3}. The order of the assertions in the assertion list does not necessitate that the tests be executed in that specific order. In particular, a PCTS.1 may undertake the header tests separately because these tests depend on the use of a compiler or other source-code translator on the development system.

It is expected that most of the developers of a PCTS.1 will make use of the C-language compiler on the implementation to undertake these tests. While this has some inherent difficulties, in that it relies on the ability of the C-language compiler to detect problems correctly, the standard developers could see no other method of accomplishing these tests. The comment that the headers may not be implemented as files makes other methods of access to the headers somewhat restricted.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

B.1.4.8 Test Methods for Macros

B.1.4.8.1 Requirements

POSIX.1 {3} requires that most elements be implemented as functions, but allows for their implementation as macros also. Access to the function is attained by the preprocessor directive `#undef` *element_name*.

Clearly, both the macro and the function are required to operate consistently, and, in cases where a macro implementation is provided, the full set of tests for the interface must be executed once for the macro and then for the underlying function. An implementation can only be considered to have passed the verification suite when both macro and function tests have succeeded.

B.1.4.12 Test Methods for Signal Functions

B.1.4.12.1 Reentrance of Interrupted Functions

B.1.4.12.1.1 Requirements

Some reentrant functions return -1 and set *errno* to [EINTR] [results from *read*(), *write*(), and *sleep*() indicating partial completion] after being interrupted. This does not mean they are affected by the call from the signal handler; they are affected by being interrupted (whether or not the signal handler calls the function again).

B.1.4.12.1.2 Recommendations

It is reasonable to expect that if the reentry of an interrupted call from the signal-handling function does not produce unexpected behavior, then interaction with calls to other functions will not cause untoward behavior. While the standard developers accepted that this may not be an exhaustive test method, it was considered too onerous on the PCTS.1 author to test that all functions behaved correctly from a signal handler.

While *kill*() is not a function that can enter a waiting state, POSIX.1 {3} states that if an unblocked signal is generated for the sending process by *kill*(), it shall be delivered before *kill*() returns. This provides the ability to test the reentrancy of the *kill*() function.

The assertion list separates those functions that enter into a waiting state from those that do not. It is not possible to ensure that a signal occurs during the processing of a function that does not enter into a waiting state, and the assertion is marked as extended for such functions.

B.1.4.12.2 Functions Interruptable by Signals

B.1.4.12.2.1 Requirements

The processing of job control signals differs from that of other signals, and the default action of these signals does not cause the function to report the interrupt but rather to stop or continue processing as if the signal had not occurred.

Separate assertions are included to verify the behavior of job control signals on functions.

B.1.4.12.3 *sigaction*() Testing

B.1.4.12.3.2 Recommendations

A failure of the *sigaction*() function can only be guaranteed for the SIGSTOP and SIGKILL functions. The use of the SIGKILL function would terminate the process, and it would not be possible to determine whether the signal-catching function had been established. Therefore the SIGSTOP function must be used in this test. This has the unfortunate side effect that this test cannot be executed on systems without job control.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

B.1.4.13 Test Methods for Files and Input/Output Primitives

B.1.4.13.1 Requirements

The requirement to test at least one file type, as opposed to all possible file types, is derived from the requirement to perform thorough testing. In some cases, when applied to files (especially pipes and FIFOs), POSIX.1 {3} clearly differentiates in the standard or its rationale as to which file types exhibit the behavior described. In other instances, no distinction is made between different types of file.

An interesting experiment may be performed by contemplating the range of file types that could be used to test 6.2.4, exec, 57(D).

B.1.4.13.2 Recommendations

Rather than listing each of the file types to be used by each assertion, the global statement in this subclause has been included. It is expected that a PCTS.1 shall test all of the statements against the regular file type but that a more thorough PCTS.1 may consider testing against other file types.

For example, this standard would be satisfied by a PCTS.1 that tests that the *link*() function works as specified on regular files. However, the definition of *link*() in POSIX.1 {3} does not differentiate between a regular file and a FIFO, and a PCTS.1 may extend its thoroughness by testing against a FIFO.

B.1.4.15 Test Methods for Terminal I/O Functions

B.1.4.15.1 Requirements

IEXTEN must be off for all assertions unless explicitly stated otherwise.

B.1.4.15.2 Recommendations

Features that drive physical conditions, the proof of which can only be ascertained by physical inspection, are invisible to the test suite. Again, terminal control has numerous examples of these features. The feature allowing the selection of the number of bits per character is largely untestable since, if a supplier does not implement any form of bit selection in hardware, the test will always pass. An obscure way to detect this particular implementation is to test for a condition that is known to fail (such as having the sender at 5 b and the receiver at 8 b) and verifying that there is a failure.

B.1.4.17 Documentation Audit

B.1.4.17.1 Requirements

The standard developers discussed the documentation audit process and its relationship to documentation assertions. The procedure decided on took documentation assertions covering the PCD in its entirety and incorporated them into the documentation audit procedures.

The purpose of the PCD.1 must be remembered when performing a documentation audit. The PCD.1 is intended to allow a person familiar with POSIX.1 {3} to know how a specific implementation may differ from another with which he or she is familiar. It should allow simple comparisons between various conforming implementations to determine possible programming changes that may need to be made. For this reason, having a reasonably similar structure is a significant factor and an important thing to check.

The standard developers realize that the documentation audit is a subjective process. Two principles should be applied when auditing a PCD:

— The principle of least astonishment (or "Why is that in this section?")

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— The principle of greatest expectation (or "Why is that not in this section?")

For information not required by a documentation assertion to be in a specific section, either or both of the above principles should be applied in determining the appropriate section(s) for that information. The person performing the documentation audit is not required to perform a qualitative assessment if the PCD wording, style, or the "best" location for a certain piece of information to appear, if its location is not specified by a documentation assertion.

B.1.4.19 Test Methods for C-Language Functions

B.1.4.19.1 Requirements

After considerable discussion, consensus could not be reached. The following represents one view.

When compliance to Common Usage C is being tested, a combination of the PCTS.1 and the PCTP.1 is used to determine final result codes. In the case where the PCTS.1 identifies that a C function is not supported by the implementation or deviates, the PCTS.1 produces an **UNRESOLVED** result code, and the PCTP.1 verifies that the conformance document declares this C function to be unimplemented. If the PCTP.1 verifies this to be true, the result code is resolved to **UNSUPPORTED**; otherwise, it resolves to **FAIL**.

POSIX.1 {3} allows for C-language functions to be omitted from a conforming implementation in a common C-language environment, subject to these omissions being detailed in the conformance document. The test method requires that a combination of the software (PCTS.1) and associated procedures (PCTP.1) be used to determine the final result. The alternative of preceding each assertion with the term "If the function is supported" and changing the assertion type to conditional was rejected by the standard developers because it was inadequate for detecting deviances in Common C routines. For example, it would be acceptable for the function declaration for *malloc*() to declare *long* if this was documented as a deviance from the C Standard {2}. The combination of PCTS.1 and PCTP.1 allows for correct resolution of such situations in a straightforward manner.

B.2 Definitions and General Requirements

B.2.2.2 General Terms

The majority of the behavioral aspects described in the General Terms clause of POSIX.1 {3} are tested in relation to specific functions. It was decided not to duplicate any tests in this section but rather to highlight any definitions that were applicable to a wide range of functions.

While it is possible for an application to make use of other characters in the generation of file names, the characters specified are the only ones that can be used by portable applications. The assertion is designed to ensure that the implementation supports at least this set of characters and, therefore, can be used for the development and execution of portable applications.

B.2.3 General Concepts

The general concepts section in POSIX.1 {3} refers to the concept of extended security controls. These extended security controls are implementation defined and may modify the behavior of certain functions defined in POSIX.1 {3}. In particular, extended security controls may prohibit access to or knowledge of objects (files, processes, etc.) on the target system.

Until the interfaces to these extended security control mechanisms are standardized (POSIX.6), a PCTS.1 may assume that it is able to execute in an environment that uses only those standard security mechanisms defined in POSIX.1 {3}. In a situation where an implementation provides only extended security mechanisms, the impact on the test method is unspecified.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.2.3.6 Pathname Resolution

When error returns from interfaces are expected, these assertions cannot be tested. The fact that an interface produces an error consistent with the correct resolution of the pathname does not prove that the pathname was resolved correctly. For example, an [EEXIST] error could have been generated by resolving to any existing file instead of the correct one, or an [EISDIR] by resolving to "." instead of "/". Hence, these parts of the general assertions for pathname resolution are untestable and are expanded into class (B) or (D) assertions. Also, all the *pathconf*() assertions are extended as there is no method to test that *pathconf*() has resolved to a particular file instead of to a different file that would produce the same return value.

B.2.5 Primitive System Data Types

There is no PID_MAX.

B.2.6 Environment Descriptions

There are no assertions for the environment variables **LANG**, **LC_***, **TERM**, and **TZ** because these variables may not exist or may take a form that has no effect on the semantics of any function defined in POSIX.1 {3}. POSIX.1 {3} places no requirement on the existence or the values associated with these environment variables, and, as such, no assertions can be developed for them.

B.2.9 Symbolic Constants

While POSIX.1 {3} does not explicitly state that the values of the elements of *lseek*() (SEEK_SET, SEEK_CUR, or SEEK_END) need to be distinct, this can be inferred from the usage of these values.

B.3.1.1 *fork*()

POSIX.1 {3} states that it is not safe for both the child and parent processes to continue processing a directory stream after a *fork*(). It is necessary to assert that the child process is able to continue processing the directory stream if no processing of this stream is undertaken in the parent.

The assertion associated with the detection of [ENOMEM] represents the concensus of the standard developers. This assertion, in combination with its required documentation assertion, should result in a testable assertion for any specific implementation.

B.3.1.2 exec

To test time for `exec` functions, run up at least 20 s on the times of a process. Check that after the `exec` function is called, these times are greater than or equal to the values for the previous process image.

Assertions 22, 23, 26, and 27 are untestable, since many implementations do not support execution of directory files, and when they do, the behavior when an attempt is made to execute a directory is unspecified.

A zero length file is used to test assertion 59. However, corrupting the length of a valid executable will almost certainly be possible on any implementation. There probably needs to be a target system support routine to achieve this.

B.3.3.1.2 Signal Generation and Delivery

To test assertion 23:

1) Generate a signal that is currently set to be blocked and whose action is to be ignored.
2) Check if the signal is pending. If it is not, then the test succeeds.
3) Unblock the signal and check that it is not delivered.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4) Generate the signal as specified in step (1) above.
5) Change the signal action to execute a signal-catching function on delivery. Check that the signal is still pending.
6) Change the signal action to ignore the signal. Check that the signal is no longer pending.

While it is expected that the signal will be discarded on generation in most implementations, it may be that some implementations retain the signal for possible future delivery. These implementations may not correctly deal with the discard of the signal when the signal action is set again to ignore.

B.3.3.2 *kill*()

POSIX.1 {3} states that *extended security controls* may impose further implementation-defined restrictions on the sending of signals. This would normally mandate, as a minimum, conditional type classifications on base assertions associated with sending signals because implementation-defined restrictions are not portably testable. A PCTS.1 uses signals to test many assertions that directly and indirectly relate to signals.

Because a portable PCTS.1 testing POSIX.1 {3} cannot be implemented that handles *extended security controls*, the standard developers agreed to write this standard without addressing the *extended security controls* issues.

B.3.3.6 *sigpending*()

As with several other functions in POSIX.1 {3}, an unsuccessful return value is defined for *sigpending*(), but no mandatory error conditions are given. In such cases, the assertion is marked as extended because it is not possible to produce a portable test.

B.5.3.4 *link*()

When a pathname assertion is tested and the assertion specifies a directory pathname, whether links to directories are supported is not applicable to the assertion. The ability to test the assertion portably requires that links to directories are supported. Hence, the conditional feature {PCD_LINK_TO_DIRECTORY} is employed as a testing constraint, not as a conditional feature.

B.5.4.1 *mkdir*()

The POSIX.1 {3} definition of an empty directory allows at most, directory entries for dot and dot-dot.

Creating a directory may cause the link count of the parent directory to be incremented. The link count bias occurs in operating systems due to the dot-dot entry created in the new directory. POSIX.1 {3} implementations that support the dot-dot concept may not actually create an entry for dot-dot and need not support the increment of the link count of the parent directory.

B.5.6.5 *chown*()

The *chown*() assertions that test the clearing of the S_ISUID and S_ISGID bits cannot be tested for being cleared if {_POSIX_CHOWN_RESTRICTED} is defined.

B.6.3.1 *close*()

Assertion 11 can be tested in the following manner:

1) *time*()
2) IO_mark_for_update operation
3) *close*()
4) *sleep*(*DELAY*)
5) *stat*()

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

B.7.1.1.3 The Controlling Terminal

Assertion 4 can be tested in the following manner:

1) Create a process and put it in its own session using *setsid*(). It now has no controlling terminal.
2) Open a terminal file with the O_NOCTTY flag set.
3) Send an interrupt from the terminal. This should raise a signal if the terminal was a controlling terminal, but not otherwise.

B.7.1.1.7 Noncanonical Mode Input Processing

To test TIME (Section 7.1.1.7 of POSIX.1 {3}), set TIME to PCTS_TIME_EPSILON, then count 10 s. The TIME should be correct within 10%. PCTS_TIME_EPSILON should be greater than or equal to 100.

B.7.1.1.11 Closing a Terminal Device File

Assertions where the conditions require "last process to close a terminal device" cannot be portably tested for a controlling terminal since a system process may have the terminal open. These assertions should be tested on a noncontrolling terminal.

B.7.1.2.2 Input Modes

POSIX.1 {3} states that the precise conditions under which START and STOP characters are transmitted are undefined. This has been taken to mean that there will be some variance between implementations.

B.7.2.1.2.1 *tcsetattr*()

B0 is a special case. Whether the system supports split baud rates or not is not relevant.

B.7.2.4 *tcsetpgrp*()

Assertion 10 can be tested in the following manner:

	Parent	Child
(a)	Establish controlling terminal file descriptor (*tty_fildes*)	
(b)	*fork*()	
(c)	*exit*()	
(d)		*tcsetpgrp*(*tty_fildes*, *getpgrp*())

The exit of the parent should disassociate the *tty_fildes* controlling terminal from the session, and the *tcsetpgrp*() in the child should then fail.

B.8.1.18 *freopen*()

Additional consideration is required for Assertion 38. Normally a call to *freopen*() closes a file descriptor and then opens one. It will not fail with {ENFILE} if too many file descriptors are already open before the call. The only way an {ENFILE} can occur is if a new file descriptor is opened by another process in between the close and open operations performed by *freopen*().

B.10.1.1 Extended `tar` Format

The test for TMAGIC of "ustar" depends on the archive-creating utility producing a file of this type.

Copyright © 2000 IEEE All Rights Reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ISO/IEC 14515-1:2000(E)
IEEE Std. 2003.1, 1992 edition (Reaff 2000)

ICS 35.060
Price based on 317 pages

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale