

# **STANDARDS PROJECT**

**Draft Standard for IEEE Draft 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]**

Sponsor  
**Portable Applications Standards Committee**  
of the  
**IEEE Computer Society**

**Abstract:** 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. Since is 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) which 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

**P2003.lb / D6**  
**Oct 1, 1998**

Copyright © 1998 by the Institute of Electrical and Electronics Engineers, Inc.  
345 East 47th Street  
New York, NY 10017, USA  
All rights reserved.

*This is an unapproved draft of a proposed IEEE Standards, subject to change.  
Permission is hereby granted for IEEE Standards Committee participants  
to reproduce this document for purposes of IEEE standardization activities.  
Permission is also granted for member bodies and technical committees of ISO and IEC  
to reproduce this document for purposes of developing a national position.  
Other entities seeking permission to reproduce this document for standardization or other  
activities, or to reproduce portions of this document for these or other uses, must contact the  
IEEE Standards Department for the appropriate license. Use of information contained in  
this unapproved draft is at your own risk.*

IEEE Standards Department  
Copyright and Permissions  
445 Hoes Lane, P.O. Box 1331  
Piscataway, NJ 08855-1331, USA  
+1 (908) 562-3800  
+1 (908) 562-1571 [FAX]

*October 1, 1998*

*SH XXXXX*

1     ***Editor's Notes***

2     This section will not appear in the final document. It is used for editorial comments concerning  
3     this draft.

4     This document is based on the merged document ISO/IEC 9945-1:1990 and P1003.4 Draft 14  
5     which resulted in the published IEEE Std 1003.1b-1993. This draft uses small numbers in the  
6     right margin to denote changes in various drafts of this document. Trivial informative (i.e., non-  
7     normative) changes and purely editorial changes such as grammar, spelling, or cross references  
8     are not diff-marked. While most sections should be correctly integrated, the rationale has had  
9     only the most rudimentary integration.

10    In this document, the FOR: clause is treated as a looping construct similar to that found in  
11    programming languages. It lists a set of functions or constants that are substituted for a *function*  
12    () or CONSTANT reference in the body of the assertion. This new structure and use will be  
13    forwarded to the 2003 working group for consideration.

14    Another convention used in this draft is the use of *PCTS\_variables*. Because of the if... Otherwise  
15    structure in POSIX.1b {3}, it was necessary to combine the use of {\_POSIX\_} compile-time  
16    symbolic constants that indicate optional facilities in POSIX.1b {3} with the case where the  
17    implementor chooses to provide some of the functions specified to be present when the option  
18    is defined. For example, an implementation that provides the *sem\_init()* function but does not  
19    provide all facilities required for the {\_POSIX\_SEMAPHORES} option to be set would require a  
20    PCTS to be configured with the constant *PCTS\_sem\_init* to be true. Note, this constant would also  
21    need to be TRUE if the {\_POSIX\_SEMAPHORES} option is set.

22    The rationale text for all the sections has been temporarily moved from Annex B and  
23    interspersed with the appropriate sections. The rationale sections are identified with the phrase  
24    "*(This subclause is not a part of P2003.1b)*" in the heading. This co-location of rationale with its  
25    accompanying text was done to encourage the Technical Reviewers to maintain the rationale  
26    text, as well as provide explanations to the reviewers and balloters. Not all of the Rationale  
27    sections have contents as of this draft. The empty sections may be partially distracting, but we  
28    feel it is imperative to keep them there to encourage the Technical Reviewers to provide  
29    rationale as needed.

30 *Please report typographical errors to:*

31 Barry Hedquist  
32 Perennial, Inc.  
33 15810 Miradero Ave, Suite 200.  
34 San Jose, CA 95127  
35 TEL: +1 (408) 347-7800  
36 FAX: +1 (408) 347-7803  
37 Email: [beh@peren.com](mailto:beh@peren.com)

38 *(Electronic mail is preferred.)*

39 The copying and distribution of IEEE balloting drafts is accomplished by the Standards Office.  
40 To report problems with reproduction of your copy, or to request additional copies of this draft,  
41 contact:

42 Anna Kaczmarek or Theresa Bien  
43 IEEE Standards Office  
44 P.O. Box 1331  
45 445 Hoes Lane  
46 Piscataway, NJ 08855-1331

47 (908) 562-3811 [Anna]  
48 (908) 562-3833 [Theresa]  
49 FAX: (908) 562-1571

50 *P2003.1b Change History*

51 This section is provided to track major changes between drafts. It was first added in Draft 3.

52 Draft 3 First complete draft of all of POSIX.1b {3} assertions. *PCTS\_* variables are  
53 defined. Subclause 1.4 was added, but is incomplete.

54 Draft 4 First Balloted Draft, revision, and more complete version of Draft 3.

55 Draft 5 First Recirculation Ballot. Contained ballot resolutions for Draft 4.

56 Draft 6 Second recirculation ballot. Contained ballot resolutions for Draft 5.

**IEEE Standards** documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE 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 the 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, the IEEE and the members of its technical 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 Standards Board  
445 Hoes Lane  
P.O. Box 1331  
Piscataway, NJ 08855-1331

IEEE Standards documents are adopted by the Institute of Electrical and Electronics Engineers without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.
---

# Contents

Introduction . . . . .	1
Organization of This Standard . . . . .	1
How to Read This Standard . . . . .	1
Related Standards Activities . . . . .	2
Section 1: General . . . . .	5
1.1 Scope . . . . .	5
1.2 Normative References . . . . .	6
1.3 Conformance . . . . .	7
1.4 Test Methods . . . . .	8
Section 2: Terminology and General Requirements . . . . .	23
2.1 Conventions . . . . .	23
2.2 Definitions . . . . .	24
2.3 General Concepts . . . . .	32
2.4 Error Numbers . . . . .	38
2.5 Primitive System Data Types . . . . .	39
2.6 Environment Description . . . . .	39
2.7 C Language Definitions . . . . .	41
2.8 Numerical Limits . . . . .	44
2.9 Symbolic Constants . . . . .	48
Section 3: Process Primitives . . . . .	52
3.1 Process Creation and Execution . . . . .	52
3.1.1 Process Creation . . . . .	52
3.1.2 Execute a File . . . . .	54
3.2 Process Termination . . . . .	57
3.2.1 Wait for Process Termination . . . . .	57
3.2.2 Terminate a Process . . . . .	57
3.3 Signals . . . . .	59
3.3.1 Signal Concepts . . . . .	59
3.3.2 Send a Signal to a Process . . . . .	68
3.3.3 Manipulate Signal Sets . . . . .	69
3.3.4 Examine and Change Signal Action . . . . .	69
3.3.5 Examine and Change Blocked Signals . . . . .	70
3.3.6 Examine Pending Signals . . . . .	71
3.3.7 Wait for a Signal . . . . .	71
3.3.8 Synchronously Accept a Signal . . . . .	71
3.3.9 Queue a Signal to a Process . . . . .	75
3.4 Timer Operations . . . . .	79
3.4.1 Schedule Alarm . . . . .	79
3.4.2 Suspend Process Execution . . . . .	79
3.4.3 Delay Process Execution . . . . .	80

Section 4: Process Environment .....	81
4.8 Configurable System Variables .....	81
4.8.1 Get Configurable System Variables .....	81
Section 5: Files and Directories .....	85
5.1 Directories .....	85
5.1.1 Format of Directory Entries .....	85
5.1.2 Directory Operations .....	85
5.2 Working Directory .....	85
5.2.1 Change Current Working Directory .....	85
5.2.2 Get Working Directory Pathname .....	86
5.3 General File Creation .....	86
5.3.1 Open a File .....	86
5.3.2 Create New File or Rewrite an Existing One .....	87
5.3.3 Set File Creation Mask .....	87
5.3.4 Link to a File .....	87
5.4 Special File Creation .....	88
5.4.1 Make a Directory .....	88
5.4.2 Make a FIFO Special File .....	88
5.5 File Removal .....	89
5.5.1 Remove Directory Entries .....	89
5.5.2 Remove a Directory .....	89
5.5.3 Rename a File .....	89
5.6 File Characteristics .....	90
5.6.1 File Characteristics: Header and Data Structure .....	90
5.6.2 Get File Status .....	91
5.6.3 Check File Accessibility .....	92
5.6.4 Change File Modes .....	92
5.6.5 Change Owner and Group of a File .....	95
5.6.6 Set File Access and Modification Times .....	96
5.6.7 Truncate a File to a Specified Length .....	96
5.7 Configurable Pathname Variables .....	98
5.7.1 Get Configurable Pathname Variables .....	98
Section 6: Input and Output Primitives .....	102
6.1 Pipes .....	102
6.1.1 Create an Inter-Process Channel .....	102
6.2 File Descriptor Manipulation .....	102
6.2.1 Duplicate an Open File Descriptor .....	102
6.3 File Descriptor Deassignment .....	103
6.3.1 Close a File .....	103
6.4 Input and Output .....	103
6.4.1 Read from a File .....	103
6.4.2 Write to a File .....	105
6.5 Control Operations on Files .....	106
6.5.1 Data Definitions for File Control Operations .....	106
6.5.2 File Control .....	106
6.5.3 Reposition Read/Write File Offset .....	109
6.6 File Synchronization .....	110
6.6.1 Synchronize the State of a File .....	110



6.6.2	Synchronize the Data of a File .....	112
6.7	Asynchronous Input and Output .....	114
6.7.1	Data Definitions for Asynchronous Input and Output .....	114
6.7.2	Asynchronous Read .....	117
6.7.3	Asynchronous Write .....	123
6.7.4	List Directed I/O .....	130
6.7.5	Retrieve Error Status of Asynchronous I/O Operation .....	141
6.7.6	Retrieve Return Status of Asynchronous I/O Operation .....	142
6.7.7	Cancel Asynchronous I/O Request .....	144
6.7.8	Wait for Asynchronous I/O Request .....	146
6.7.9	Asynchronous File Synchronization .....	149
Section 7: Device- and Class- Specific Functions .....		156
Section 8: Language-Specific Services for the C Programming Language .....		157
8.2.2	Open a Stream on a File Descriptor .....	157
Section 9: System Databases .....		158
Section 10: Data Interchange Format .....		159
Section 11: Synchronization .....		160
11.1	Semaphore Characteristics .....	160
11.2	Semaphore Functions .....	161
11.2.1	Initialize an Unnamed Semaphore .....	161
11.2.2	Destroy an Unnamed Semaphore .....	164
11.2.3	Initialize/Open a Named Semaphore .....	165
11.2.4	Close a Named Semaphore .....	171
11.2.5	Remove a Named Semaphore .....	173
11.2.6	Lock a Semaphore .....	175
11.2.7	Unlock a Semaphore .....	183
11.2.8	Get the Value of a Semaphore .....	187
Section 12: Memory Management .....		191
12.1	Memory Locking Functions .....	194
12.1.1	Lock/Unlock the Address Space of a Process .....	194
12.1.2	Lock/Unlock a Range of Process Address Space .....	198
12.2	Memory Mapping Functions .....	203
12.2.1	Map Process Address to a Memory Object .....	203
12.2.3	Change Memory Protection .....	215
12.2.4	Memory Object Synchronization .....	220
12.3	Shared Memory Functions .....	224
12.3.1	Open a Shared Memory Object .....	224
12.3.2	Remove a Shared Memory Object .....	231
Section 13: Execution Scheduling .....		235
13.1	Scheduling Parameters .....	235
13.2	Scheduling Policies .....	235

13.2.1	SCHED_FIFO .....	236
13.2.2	SCHED_RR .....	238
13.2.3	SCHED_OTHER .....	238
13.3	Process Scheduling Functions .....	239
13.3.1	Set Scheduling Parameters .....	239
13.3.2	Get Scheduling Parameters .....	242
13.3.3	Set Scheduling Policy and Scheduling Parameters .....	244
13.3.4	Get Scheduling Policy .....	248
13.3.6	Get Scheduling Parameter Limits .....	250
Section 14:	Clocks and Timers .....	255
14.1	Data Definitions for Clocks and Timers .....	255
14.1.1	Time Value Specification Structures .....	255
14.1.2	Timer Event Notification Control Block .....	256
14.1.3	Type Definitions .....	256
14.1.4	Manifest Constants .....	256
14.2	Clocks and Timer Functions .....	257
14.2.1	Clocks .....	257
14.2.2	Create a Per-Process Timer .....	263
14.2.3	Delete a Per-Process Timer .....	267
14.2.4	Per-Process Timers .....	269
14.2.5	High Resolution Sleep .....	275
Section 15:	Message Passing .....	280
15.1	Data Definitions for Message Queues .....	280
15.1.1	Data Structures .....	280
15.2	Message Passing Functions .....	281
15.2.1	Open a Message Queue .....	281
15.2.2	Close a Message Queue .....	289
15.2.3	Remove a Message Queue .....	291
15.2.4	Send a Message to a Message Queue .....	293
15.2.5	Receive a Message From a Message Queue .....	295
15.2.6	Notify Process that a Message is Available on a Queue .....	298
15.2.7	Set Message Queue Attributes .....	300
15.2.8	Get Message Queue Attributes .....	302

## Annex A

(normative)

<b>Conforming Test Results</b> .....	304
A.1 General .....	304
A.2 Terminology and General Requirements .....	304
A.3 Process Primitives .....	306
A.4 Process Environment .....	314
A.5 Files and Directories .....	314
A.6 Input and Output Primitives .....	318
A.7 Device- and Class-Specific Functions .....	326
A.8 Language-Specific Services for the C Programming Language .....	326
A.9 System Databases .....	326
A.10 Data Interchange Format .....	327

A.11	Synchronization	327
A.12	Memory Management	331
A.13	Execution Scheduling	338
A.14	Clocks and Timers	342
A.15	Message Passing	347

Annex B

(informative)

<b>Bibliography</b>	353
B.1 Related Open Systems Standards	353
B.2 Other Standards	355
B.3 Historical Documentation and Introductory Texts	355
B.4 Other Sources of Information	357

Identifier Index	359
------------------	-----

Alphabetic Topical Index	363
--------------------------	-----

TABLES

Table 1-1 – PCTS Symbols and Values	10
Table 2-1 – Minimum Values	45
Table 2-2 – Run-Time Invariant Values (Possibly Indeterminate)	46
Table 2-3 – Maximum Values	48
<b>Table 2-4 – Compile-Time Symbolic Constants</b>	49
Table 2-5 – Execution-Time Symbolic Constants	51
Table 3-1 – Required Signals	59
Table 3-2 – Job Control Signals	59
Table 3-3 – Memory Protection Signals	60
Table 4-2 – Configurable System Variables	83



# Introduction

(This Introduction is not a normative part of P2003.1b, Draft Standard for Draft Standard for Information Technology – Test Methods Specifications for Measuring Conformance to POSIX.1b)

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

## 9 **Organization of This Standard**

10 This document is organized into five parts:

- 11 (1) Statement of scope, normative references, conformance requirements, and test methods  
12 (Section 1)
- 13 (2) Conventions and definitions (Section 2)
- 14 (3) Assertions to test POSIX.1b {3} (Sections 2 through 15)
- 15 (4) Conforming test results (Annex A)
- 16 (5) Bibliography (Annex B)

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

## 19 **How to Read This Standard**

20 This document is organized using the same section numbering as POSIX.1b {3} in order to  
21 facilitate finding the testing requirements for a particular element of POSIX.1b {3}. Subclause  
22 1.4 has been added to the structure of this document in order to keep the section numbering  
23 consistent with POSIX.1b {3} and to provide a place to describe features relevant to the test  
24 methods. It is strongly recommended that the reader review subclause 1.4 after reading this  
25 introduction and subclauses 1.1 and 1.2. Where possible, this document tries to use the same  
26 assertion numbering that was used as IEEE Std 2003.1-1992 {4}. Where there is no assertion in  
27 this document that corresponds to an assertion in IEEE Std 2003.1-1992 {4}, this document skips  
28 over the assertion number. Where more than one assertion in this document take the place of a  
29 single assertion in IEEE Std 2003.1-1992 {4}, the assertions have been numbered with a  
30 fractional part.

31 **Related Standards Activities**

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

34 The following areas are under active consideration at this time, or are expected to become active  
35 in the near future: <sup>1)</sup>

- 36 (1) Language-independent service descriptions of this standard
- 37 (2) C, Ada, and FORTRAN language bindings to (1)
- 38 (3) Shell and utility facilities
- 39 (4) Verification testing methods
- 40 (5) Multi-threaded process facilities
- 41 (6) Secure/Trusted system considerations
- 42 (7) Network interface facilities
- 43 (8) System administration
- 44 (9) Graphical user interfaces
- 45 (10) Profiles describing application- or user-specific combinations of Open Systems  
46 standards for: supercomputing, multiprocessor, and batch extensions; transaction  
47 processing; realtime systems; and multiuser systems based on historical models.
- 48 (11) An overall guide to POSIX-based or related Open Systems standards and profiles.

49 Extensions are approved as "amendments" or "revisions" to this document, following IEEE  
50 Standards procedures.

51 Approved amendments are published separately until the full document is reprinted and such  
52 amendments are incorporated in their proper positions.

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

---

1)

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

61 IEEE Std P2003.1b-199X was prepared by the 2003 working group in a breakout group focused  
62 on developing the 2003.1b standard, sponsored by the Portable Applications Standards  
63 Committee of the IEEE Computer Society. At the time this standard was approved, the  
64 membership of the 2003.1b breakout groups was as follows:

65 **Portable Applications Standards Committee**

66 Chair: Lowell Johnson  
67 Vice Chair: Joe Gwinn  
68 Secretary: Nick Stoughton  
69 Functional Chairs: Andrew Josey  
70 Jay Ashford  
71 Curtis Royster  
72 Jason Zions

73 **2003 Working Group Officials**

74 Chair: Barry Hedquist  
75 Roger Martin (retired)  
76 Vice Chair: John Davies  
77 Ken Thomas (2003.1b)  
78 Editor: Bruce Weiner (primary)  
79 Jeffrey S. Haemer (supporting)  
80 Secretary: Keith Stobie (1994)

81 **Technical Reviewers**

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

84 **2003.1b Working Group**

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

87 The following people were members of the 2003.1b Balloting Group that approved the standard  
88 for submission to the IEEE Standards Board:

89 Khaled Al-Ali Lowell Johnson William R. Smith  
90 Andy Bihain Arkady Kanevsky Kenneth G. Thomas  
91 Susan Corwin Roger Martin Bruce Weiner  
92 Steven J. Dovitch James Oblinger Andrew Wheeler  
93 Michel Gien Gerald Powell Alex White  
94 Patrick Hebert Curtis Royster John Zolnowsky  
95 Barry Hedquist Andrew Schoka

When the IEEE Standards Board approved this standard on *<date to be provided>*, it had the following membership:

(to be pasted in by IEEE)



# Draft Standard for Information Technology - Test Methods Specifications for Measuring Conformance to POSIX.1b

## Section 1: General

### 86 1.1 Scope

87 This standard defines the test method specifications for measuring conformance of an implementation to POSIX.1b  
 88 {3}.<sup>2)</sup> The primary test method specification for measuring conformance to POSIX.1b {3} are assertions.  
 89 Conformance to POSIX.1b {3} requires conformance to IEEE Std 1003.1b-1993 as it has been modified by POSIX.1b  
 90 {3}. Therefore, the test method specifications for POSIX.1b {3} assume that the test method specifications of IEEE  
 91 Std 2003.1-1992 {4} apply, except as modified by this standard.

92 This standard is intended for use by

- 93 (1) Developers of POSIX.1b {3} test methods;
- 94 (2) Implementors of POSIX.1b {3} implementations;
- 95 (3) Application writers for POSIX.1b {3} conforming implementations;
- 96 (4) POSIX.1b {3} testing laboratories; and
- 97 (5) Others interested in validating the conformance of a vendor-claimed POSIX.1b {3} implementation.

---

2)

15 The numbers in braces correspond to those of the references in §1.2.

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

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

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

24 P2003.1b defines a means of measuring conformance to the POSIX.1b {3} technical specifications. Any question  
25 of interpretation of technical specifications arising from the use of this standard is a question of interpretation of  
26 POSIX.1b {3}.

## 25 1.2 Normative References

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

31 {1} ISO/IEC 646: 1991, <sup>3)</sup> *Information processing – ISO 7-bit coded character set for information interchange.*

32 {2} ISO/IEC 9899: 1990, *Programming languages – C.*

33 {3} IEEE Std 1003.1b-1993, *IEEE Standard for Information Technology – POSIX – Part 1: System Application  
34 Program Interface (API) – Amendment 1: Realtime Extension [C Language].*

35 {4} IEEE Std 2003.1-1992, *IEEE Standard for Information Technology – Test Methods for Measuring Conformance  
36 to POSIX – Part 1: System Interfaces.*

37 {5} IEEE Std P2003-1997, *IEEE Standard for Information Technology – Requirements and Guidelines for Test  
38 Methods Specifications and Test Method Implementations for Measuring Conformance to POSIX Standards.*

---

3)

42 ISO/IEC documents can be obtained from the ISO office, 1, rue de Varembe, Case Postale 56, CH-  
43 1211, Genève 20, Switzerland/Suisse.

## 42 1.3 Conformance

### 43 1.3.1 Implementation Conformance

#### 44 1.3.1.1 Requirements

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

#### 46 1.3.1.2 Documentation

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

#### 48 1.3.1.3 Conforming Implementation Options

49 There are no requirements for conforming implementations in this subclause.

### 50 1.3.2 Application Conformance

51 There are no requirements for conforming implementations in this subclause.

### 52 1.3.3 Language -Dependent Services for the C Programming Language

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

#### 54 1.3.3.1 Types of Conformance

55 There are no requirements for conforming implementations in this subclause.

#### 56 1.3.3.2 C Standard Language-Dependent System Support

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

#### 58 1.3.3.3 Common-Usage C Language-Dependent System Support

59 (IEEE Std 2003.1-1992 {4} DGA01

60 *UNUSED*

61 *M\_GD\_CommonC\_diffs (function) =*

62 **FOR:** *function ()*

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

64 **TEST:** Subclause 8.1 of the PCD.1b contains the details of all differences between the  
65 interface *function ()* provided and that required by the C Standard {2}.

66 **ELSE** *NO\_OPTION*

67 **GD\_CommonC\_diffs**

68 **FOR:** *function ()*

69 *M\_GD\_CommonC\_diffs (function)*

70 *Conformance for conformance: PASS, NO\_OPTION*

#### 71 1.3.4 Other C Language-Related Specifications

72 (IEEE Std 2003.1-1992 {4} GA01

73 *UNUSED*

74 *M\_GA\_macro\_args (function; header1; header2; header3; header4) =*

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

76 **SETUP:** The headers <header1>, <header2>, <header3>, and <header 4> are  
77 included.

78                   **TEST:**     When the macro *function* () is invoked with the correct argument types (or  
79                                   compatible argument types in the case that C Standard {2} support is provided), the  
80                                   macro evaluates its arguments only once, fully protected by parentheses when  
81                                   necessary, and protects its result value with extra parentheses when necessary.  
82                   **ELSE** *NO\_OPTION*  
83  
84   **GA\_macro\_args**  
85                   **FOR:**     All interfaces except *abort* (), *assert* (), *getc* (), *putc* (), *setjmp* (), *sig-setjmp* (), and *tzset*  
86                                   ().  
87                   **M\_GA\_macro\_args** (*function*; *header1*; *header2*; *header3*; *header4*)  
88                   *Conformance for conformance: PASS, NO\_OPTION*

89   **M\_GA\_macro\_result\_decl** (*function\_type*; *function*; *header1*; *header2*; *header3*; *header4*) =  
90                   **IF** the interface *function* () is defined as a macro **THEN**  
91                                   **SETUP:**    The headers <*header1*>, <*header2*>, <*header3*>, and <*header4*>  
92                                   are included.  
93                                   **TEST:**     When the macro *function* () is invoked with the correct argument types (or  
94                                   compatible argument types in the case that C Standard {2} support is provided), it  
95                                   expands to an expression with the result type *function\_type*.  
96                   **ELSE** *NO\_OPTION*

97   **GA\_macro\_result\_decl**  
98                   **FOR:**     All functions implemented as macros  
99                   **M\_GA\_macro\_result\_decl** (*function*; *header1*; *header2*; *header3*; *header4*)  
100                   *Conformance for conformance: PASS, NO\_OPTION*

### 101   **1.3.5 Other Language-Related Specifications**

102   There are no requirements for conforming implementation in this subclause.

## 103   **1.4 Test Methods**

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

### 106   **1.4.1 Conventions**

107   The conventions used in specifying the assertions in this standard follow those of IEEE Draft Std P2003-199X {5}  
108   with the extensions described below.

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

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

118   **(IEEE Std 2003.1-1992 {4} 03**  
119        *UNUSED*

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

122 Because the syntactic conventions used by this standard are those of IEEE Draft Std P2003-199X {5}, general  
 123 assertions and general documentation assertions are rewritten in this document, even if they are also found in IEEE  
 124 Std 2003.1-1992 {4}.

#### 125 **1.4.1.1 Phrases**

126 The following are phrases that are commonly used in this standard; they have the indicated meaning.

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

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

132 *There are no requirements for conforming implementations in this subclause.*

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

#### 135 **1.4.1.2 Macros**

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

```
141 M_GA_stdC_proto_decl(func_type; func_name; param1, param2, ..., header1; header2; header3; header 4)=
142 IF standard THEN
143     SETUP: The headers <header1>, <header2>, <header3>, and <header
144         4> are included.
145     TEST: The function prototype func_type func_name(param1, param2, ...) is declared.
146     ELSE NO_OPTION
```

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

#### 149 **1.4.1.3 Cross References**

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

### 153 **1.4.2 Abbreviations**

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

155 **1.4.2.1 C Standard:** ISO/IEC 9899, *Programming languages-C* {2}.

156 **1.4.2.2 IRV:** The International Reference Version coded character set described in ISO/IEC 646 {1}.

157 **1.4.2.3 POSIX.1b:** IEEE Std 1003.1b-1993 {3}.

158 **1.4.2.4 POSIX.1tm:** IEEE Std 2003.1-1992 {4}.

159 **1.4.2.5 POSIX.tm:** IEEE Draft Std P2003.199X {5}.

160 **1.4.2.6 PCD.1b:** POSIX Conformance Document as specified in subclause 1.3.1.2 of IEEE Std 1003.1b-1993  
 161 as amended by POSIX.1b {3}.

162 **1.4.2.7 IUT:** Implementation under test, the software that implements POSIX.1b {3}.

### 163 1.4.3 PCTS Variables

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

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

172 Some of the interface variables have the string "\_GAP\_" or "\_RAP\_" in them. These strings indicate that the  
 173 interface variable should be TRUE if there is a way to "get appropriate privilege" and to "release appropriate  
 174 privilege," respectively.

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

177 **Table 1-1 – PCTS Symbols and Values**

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

	Symbol	Value
203 204 205 206	<i>PCTS_DETECT_AIO_ERROR_AIOCBP</i>	<b>TRUE</b> if the implementation can detect that an <i>aiochp</i> argument does not refer to an asynchronous I/O operation whose return status has not yet been retrieved
207 208 209 210	<i>PCTS_DETECT_ENOSPC</i>	<b>TRUE</b> if the implementation can detect that an out of space condition occurs when writing to a device containing the specified file, else <b>FALSE</b> .
211 212 213 214 215	<i>PCTS_DETECT_EPERM_CLOCK_SETTIME</i>	<b>TRUE</b> if the implementation can detect that a process does not have the appropriate privileges to set the specified clock in a call to the <i>clock_settime</i> () function, else <b>FALSE</b>
216 217 218 219	<i>PCTS_DETECT_INVALID_FLAGS_MMAP</i>	<b>TRUE</b> if the implementation can detect that the arguments of <i>addr</i> or <i>off</i> are not multiples of {PAGESIZE} in a call to the <i>mmap</i> () function, else <b>FALSE</b> .
220 221 222 223	<i>PCTS_DETECT_LOCKABLE_MEMORY_LIMIT_MLOCK</i>	<b>TRUE</b> 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 <b>FALSE</b> .
224 225 226 227 228 229	<i>PCTS_DETECT_LOCKABLE_MEMORY_LIMIT_MLOCKALL</i>	<b>TRUE</b> 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 <b>FALSE</b> .
230 231 232 233	<i>PCTS_DETECT_MESSAGE_DATA_CORRUPTION</i>	<b>TRUE</b> if the implementation can detect that a data corruption problem with a message in a message queue, else <b>FALSE</b> .
234 235 236 237	<i>PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE</i>	<b>TRUE</b> if the implementation can detect that the <i>addr</i> argument is not a multiple of the page size {PAGESIZE}, else <b>FALSE</b> .
238 239 240 241 242	<i>PCTS_DETECT_NO_AP</i>	<b>TRUE</b> if the implementation can detect that the calling process does not have appropriate privilege to perform the requested operation, else <b>FALSE</b> . Used by the <i>mlock</i> () and <i>mlockall</i> () functions.

	Symbol	Value
243 244 245 246 247	<i>PCTS_EINVAL_fchmod</i>	<b>TRUE</b> if the implementation can detect that the <i>files</i> argument refers to a pipe and the implementation disallows execution of <i>fchmod()</i> on a pipe, else <b>FALSE</b> .
248 249 250 251	<i>PCTS_EXTEND_ON_ftruncate</i>	<b>TRUE</b> if the implementation extends a file to the <i>length</i> specified in a call to the <i>ftruncate()</i> function if the file previously was smaller than this size, else <b>FALSE</b> .
252 253 254	<i>PCTS_GAP_mlock</i>	<b>TRUE</b> if a process can get the appropriate privilege to lock process memory with <i>mlock()</i> , else <b>FALSE</b> .
255 256 257	<i>PCTS_GAP_mlockall</i>	<b>TRUE</b> if a process can get the appropriate privilege to lock process memory with <i>mlockall()</i> , else <b>FALSE</b> .
258 259 260	<i>PCTS_GAP_MODES_fchmod</i>	<b>TRUE</b> if a process can get the appropriate privilege to change the file permission bits of a file using <i>fchmod()</i> , else <b>FALSE</b> .
261 262 263 264	<i>PCTS_GAP_mq_open</i>	<b>TRUE</b> if a process can get the appropriate privilege to send and receive messages in the message queue specified in <i>mq_open()</i> , else <b>FALSE</b> .
265 266 267	<i>PCTS_GAP_sem_init</i>	<b>TRUE</b> if a process can get the appropriate privilege to initialize a semaphore using <i>sem-init()</i> , else <b>FALSE</b> .
268 269 270 271 272 273	<i>PCTS_GAP_SGID_fchmod</i>	<b>TRUE</b> 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 <i>S_ISGID</i> file permission bits of the file using <i>fchmod()</i> , else <b>FALSE</b> .
274 275 276 277	<i>PCTS_GAP_sigqueue</i>	<b>TRUE</b> if a process can get the appropriate privilege to change send a signal to the receiving process using <i>sigqueue()</i> , else <b>FALSE</b> .
278 279 280 281 282 283	<i>PCTS_GAP_SUID_fchmod</i>	<b>TRUE</b> 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 <i>S_ISUID</i> file permission bits of the file using <i>fchmod()</i> , else <b>FALSE</b> .
284 285 286	<i>PCTS_GAP_clock_settime</i>	<b>TRUE</b> if a process can get the appropriate privilege to set a particular clock using <i>clock-settime()</i> , else <b>FALSE</b> .



	Symbol	Value
287 288 289 290	<i>PCTS_GAP_sched_setparam</i>	<b>TRUE</b> if a process can get the appropriate privilege to set its own scheduling parameters or those of another using <i>sched_setparam()</i> , else <b>FALSE</b> .
291 292 293 294	<i>PCTS_GAP_sched_setscheduler</i>	<b>TRUE</b> if a process can get the appropriate privilege to change the scheduling parameters of another process or itself using <i>sched_setscheduler()</i> , else <b>FALSE</b> .
295 296 297	<i>PCTS_GTI_DEVICE</i>	<b>TRUE</b> if the implementation provides device types that support the General Terminal Interface, else <b>FALSE</b> .
298 299	<i>PCTS_INVALID_SIGNAL</i>	<b>TRUE</b> if the implementation has an invalid signal number, else <b>FALSE</b> .
300 301 302	<i>PCTS_MAP_FIXED</i>	<b>TRUE</b> if the implementation supports the use of the <i>MAP_FIXED</i> mode of memory mapping, else <b>FALSE</b> .
303 304 305 306	<i>PCTS_MAP_PRIVATE</i>	<b>TRUE</b> if the implementation supports the facilities indicated by the <i>MAP_PRIVATE</i> flat defined in the <code>&lt;sys/mman.h&gt;</code> else <b>FALSE</b> .
307 308 309 310 311 312 313 314 315	<i>PCTS_MORE_SA_SIGINFO_SIGNALS</i>	<b>TRUE</b> if the implementation supports the setting of the <i>si_code</i> member of the <i>siginfo_t</i> structure by means other than calling <i>kill()</i> , <i>raise()</i> , and <i>abort()</i> (if they set <i>si_code</i> to <i>SI_USER</i> ), <i>sigqueue()</i> , or <i>timer_settime()</i> or by completion of an asynchronous I/O request or by the arrival of a message on an empty message queue, else <b>FALSE</b> .
316 317	<i>PCTS_MQ_AS_FILE_TYPE</i>	<b>TRUE</b> if the implementation supports message queues as file type, else <b>FALSE</b> .
318 319	<i>PCTS_MQ_OPEN_MAX</i>	The lesser of { <i>MP_OPEN_MAX</i> }, as obtained from <i>sysconf()</i> , and 256.
320 321 322	<i>PCTS_MULTIPLE_OF_PAGESIZE</i>	<b>TRUE</b> if the implementation requires that <i>addr</i> be a multiple of the page size, { <i>PAGESIZE</i> }, else <b>FALSE</b> .
323 324	<i>PCTS_NAME_MAX</i>	The lesser of { <i>NAME_MAX</i> }, as obtained from <i>pathconf()</i> , and 2048.
325 326 327	<i>PCTS_NO_SYNC_IO_FILE</i>	<b>TRUE</b> if the implementation has file types for which synchronized I/O is not supported, else <b>FALSE</b> .

	<b>Symbol</b>	<b>Value</b>
328 329	<i>PCTS_OPEN_MAX</i>	The lesser of {OPEN_MAX}, as obtained from <i>sysconf()</i> , and 256.
330 331	<i>PCTS_PATH_MAX</i>	The lesser of {PATH_MAX}, as obtained from <i>sysconf()</i> , and 4096.
332 333	<i>PCTS_PIPE_BUF</i>	The lesser of {PIPE_BUF}, as obtained from <i>sysconf()</i> , and 32767.
334 335 336	<i>PCTS_RAP_mlock</i>	<b>TRUE</b> if the implementation supports releasing appropriate privilege for <i>mlock()</i> , else <b>FALSE</b> .
337 338 339	<i>PCTS_RAP_mlockall</i>	<b>TRUE</b> if the implementation supports releasing appropriate privilege for <i>mlockall()</i> , else <b>FALSE</b> .
340 341 342	<i>PCTS_RAP_mq_open</i>	<b>TRUE</b> if the implementation supports releasing appropriate privilege for <i>mq_open()</i> , else <b>FALSE</b> .
343 344 345	<i>PCTS_RAP_sem_init</i>	<b>TRUE</b> if the implementation supports releasing appropriate privileges for <i>sem_init()</i> , else <b>FALSE</b> .
346 347 348 349	<i>PCTS_RAP_SGID_chmod</i>	<b>TRUE</b> if the implementation supports releasing appropriate privilege for managing the S_ISGID bit in a call to <i>chmod()</i> , else <b>FALSE</b> .
350 351 352 353	<i>PCTS_RAP_SGID_fchmod</i>	<b>TRUE</b> if the implementation supports releasing appropriate privilege for managing the S_ISUID bit in a call to <i>fchmod()</i> , else <b>FALSE</b> .
354 355 356	<i>PCTS_RAP_sigqueue</i>	<b>TRUE</b> if the implementation supports releasing appropriate privilege for <i>sigqueue()</i> , else <b>FALSE</b> .
357 358 359	<i>PCTS_RAP_clock_settime</i>	<b>TRUE</b> if the implementation supports releasing appropriate privilege for <i>clock-settime()</i> , else <b>FALSE</b> .
360 361 362	<i>PCTS_RAP_sched_setparam</i>	<b>TRUE</b> if the implementation supports releasing appropriate privilege for <i>sched-setparam()</i> , else <b>FALSE</b> .
363 364	<i>PCTS_ROFS</i>	<b>TRUE</b> if the implementation supports read-only file systems, else <b>FALSE</b> .
365 366 367	<i>PCTS_SEM_EBUSY</i>	<b>TRUE</b> if the implementation supports the detection of the [EBUSY] error condition for semaphores, else <b>FALSE</b> .

	Symbol	Value
368 369 370	<i>PCTS_SEM_INVALID</i>	<b>TRUE</b> if the implementation supports a way to obtain an invalid semaphore, else <b>FALSE</b> .
371 372	<i>PCTS_SEM_IS_FD</i>	<b>TRUE</b> if the implementation supports semaphores as a file type, else <b>FALSE</b> .
373 374	<i>PCTS_SEM_NSEMS_MAX</i>	The lesser of {SEM_SEMS_MAX}, as obtained from <i>sysconf()</i> , and 1024.
375 376 377	<i>PCTS_SHM_AS_FILE_TYPE</i>	<b>TRUE</b> if the implementation supports shared memory objects as a distinct file type, else <b>FALSE</b> .
378 379	<i>PCTS_SIGQUEUE_MAX</i>	The lesser of {SIGQUEUE_MAX}, as obtained from <i>sysconf()</i> , and 64.
380 381 382 383 384	<i>PCTS_SIGTIMEDWAIT_VALUE</i>	<b>TRUE</b> if the implementation can detect when the <i>sigtimedwait()</i> function is called with a <i>timeout</i> argument specifying a <i>tv_nsec</i> value less than zero or greater than or equal to 1000 million, else <b>FALSE</b> .
385 386	<i>PCTS_TIMER_MAX</i>	The lesser of {TIMER_MAX}, as obtained from <i>sysconf()</i> , and 256.
387 388	<i>PCTS_UNSUPPORTED_SIGNAL</i>	<b>TRUE</b> if the implementation has an unsupported signal number else <b>FALSE</b> .
389 390 391 392 393	<i>PCTS_aio_cancel</i>	<b>TRUE</b> if <i>_POSIX_ASYNCHRONOUS_IO</i> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
394 395 396 397	<i>PCTS_aio_error</i>	<b>TRUE</b> if <i>_POSIX_ASYNCHRONOUS_IO</i> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
398 399 400 401 402 403	<i>PCTS_aio_fsync</i>	<b>TRUE</b> if <i>_POSIX_ASYNCHRONOUS_IO</i> and <i>_POSIX_SYNCHRONIZED_IO</i> are defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
404 405 406 407 408	<i>PCTS_aio_read</i>	<b>TRUE</b> if <i>_POSIX_ASYNCHRONOUS_IO</i> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .

	Symbol	Value
409 410 411 412	<i>PCTS_aio_return</i>	<b>TRUE</b> if <code>_POSIX_ASYNCHRONOUS_IO</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
413 414 415 416	<i>PCTS_aio_suspend</i>	<b>TRUE</b> if <code>_POSIX_ASYNCHRONOUS_IO</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
417 418 419 420	<i>PCTS_aio_write</i>	<b>TRUE</b> if <code>_POSIX_ASYNCHRONOUS_IO</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
421 422 423	<i>PCTS_clock_getres</i>	<b>TRUE</b> if <code>POSIX_TIMERS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
424 425 426	<i>PCTS_clock_gettime</i>	<b>TRUE</b> if <code>_POSIX_TIMERS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
427 428 429	<i>PCTS_clock_settime</i>	<b>TRUE</b> if <code>_POSIX_TIMERS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
430 431 432 433	<i>PCTS_fchmod</i>	<b>TRUE</b> if <code>_POSIX_MAPPED_FILES</code> or <code>_POSIX_SHARED_MEMORY_OBJECTS</code> are defined or the implementation supports the function as described in POSIX.1b{3}, else <b>FALSE</b> .
434 435 436 437	<i>PCTS_fdatasync</i>	<b>TRUE</b> if <code>_POSIX_SYNCHRONIZED_IO</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
438 439 440 441	<i>PCTS_fsync</i>	<b>TRUE</b> if <code>_POSIX_ASYNCHRONOUS_IO</code> and <code>_POSIX_SYNCHRONIZED_IO</code> are defined or the implementation supports the function as described in POSIX.1b{3}, else <b>FALSE</b> .
442 443 444 445 446	<i>PCTS_ftruncate</i>	<b>TRUE</b> if <code>_POSIX_MAPPED_FILES</code> or <code>_POSIX_SHARED_MEMORY_OBJECTS</code> are defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
447 448 449 450	<i>PCTS_get_priority_max</i>	<b>TRUE</b> if <code>_POSIX_PRIORITY_SCHEDULING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .

	<b>Symbol</b>	<b>Value</b>
451 452 453 454	<i>PCTS_get_priority_min</i>	<b>TRUE</b> if <code>_POSIX_PRIORITY_SCHEDULING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
455 456 457 458	<i>PCTS_lio_listio</i>	<b>TRUE</b> if <code>_POSIX_ASYNCHRONOUS_IO</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
459 460 461 462	<i>PCTS_mlock</i>	<b>TRUE</b> if <code>_POSIX_MEMLOCK_RANGE_</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
463 464 465 466	<i>PCTS_mlockall</i>	<b>TRUE</b> if <code>_POSIX_MEMLOCK</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
467 468 469 470 471	<i>PCTS_mmap</i>	<b>TRUE</b> if <code>_POSIX_MAPPED_FILES</code> or <code>_POSIX_SHARED_MEMORY_OBJECTS</code> are defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
472 473 474 475	<i>PCTS_mprotect</i>	<b>TRUE</b> if <code>_POSIX_MEMORY_PROTECTION</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
476 477 478 479	<i>PCTS_mq_close</i>	<b>TRUE</b> if <code>_POSIX_MESSAGE_PASSING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
480 481 482 483	<i>PCTS_mq_getattr</i>	<b>TRUE</b> if <code>_POSIX_MESSAGE_PASSING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
484 485 486 487 488	<i>PCTS_mq_notify</i>	<b>TRUE</b> if <code>_POSIX_MESSAGE_PASSING</code> and <code>_POSIX_REALTIME_SIGNALS</code> are defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
489 490 491 492	<i>PCTS_mq_open</i>	<b>TRUE</b> if <code>_POSIX_MESSAGE_PASSING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .

	Symbol	Value
493 494 495 496	<i>PCTS_mq_receive</i>	<b>TRUE</b> if <code>_POSIX_MESSAGE_PASSING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
497 498 499 500	<i>PCTS_mq_send</i>	<b>TRUE</b> if <code>_POSIX_MESSAGE_PASSING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
501 502 503 504	<i>PCTS_mq_setattr</i>	<b>TRUE</b> if <code>_POSIX_MESSAGE_PASSING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
505 506 507 508	<i>PCTS_mq_unlink</i>	<b>TRUE</b> if <code>_POSIX_MESSAGE_PASSING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
509 510 511 512 513	<i>PCTS_msync</i>	<b>TRUE</b> if <code>_POSIX_MAPPED_FILES</code> and <code>_POSIX_SYNCHRONIZED_IO</code> are defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
514 515 516	<i>PCTS_msync_storage</i>	<b>TRUE</b> if the system under test has secondary storage to which <code>msync()</code> can synchronize pages, else <b>FALSE</b> .
517 518 519 520	<i>PCTS_munlock</i>	<b>TRUE</b> if <code>_POSIX_MEMLOCK_RANGE</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
521 522 523 524	<i>PCTS_munlockall</i>	<b>TRUE</b> if <code>_POSIX_MEMLOCK</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
525 526 527 528 529	<i>PCTS_munmap</i>	<b>TRUE</b> if <code>_POSIX_MAPPED_FILES</code> or <code>_POSIX_SHARED_MEMORY_OBJECTS</code> are defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
530 531 532	<i>PCTS_nanosleep</i>	<b>TRUE</b> if <code>_POSIX_TIMERS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
533 534	<i>PCTS_read</i>	<b>TRUE</b> if the implementation supports the <code>read()</code> , function (Always <b>TRUE</b> ).

	<b>Symbol</b>	<b>Value</b>
535 536 537 538 539	<i>PCTS_sched_get_priority_max</i>	<b>TRUE</b> if <code>_POSIX_PRIORITY_SCHEDULING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
540 541 542 543	<i>PCTS_sched_get_priority_min</i>	<b>TRUE</b> if <code>_POSIX_PRIORITY_SCHEDULING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
544 545 546 547	<i>PCTS_sched_getparam</i>	<b>TRUE</b> if <code>_POSIX_PRIORITY_SCHEDULING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
548 549 550 551	<i>PCTS_sched_getscheduler</i>	<b>TRUE</b> if <code>_POSIX_PRIORITY_SCHEDULING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
552 553 554 555	<i>PCTS_sched_rr_get_interval</i>	<b>TRUE</b> if <code>_POSIX_PRIORITY_SCHEDULING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
556 557 558 559	<i>PCTS_sched_setparam</i>	<b>TRUE</b> if <code>_POSIX_PRIORITY_SCHEDULING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
560 561 562 563	<i>PCTS_sched_setscheduler</i>	<b>TRUE</b> if <code>_POSIX_PRIORITY_SCHEDULING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
564 565 566 567	<i>PCTS_sched_yield</i>	<b>TRUE</b> if <code>_POSIX_PRIORITY_SCHEDULING</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
568 569 570 571	<i>PCTS_sem_close</i>	<b>TRUE</b> if <code>_POSIX_SEMAPHORES</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
572 573 574 575	<i>PCTS_sem_destroy</i>	<b>TRUE</b> if <code>_POSIX_SEMAPHORES</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .

	Symbol	Value
576 577 578 579	<i>PCTS_sem_getvalue</i>	<b>TRUE</b> if <code>_POSIX_SEMAPHORES</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
580 581 582 583	<i>PCTS_sem_init</i>	<b>TRUE</b> if <code>_POSIX_SEMAPHORES</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
584 585 586 587	<i>PCTS_sem_open</i>	<b>TRUE</b> if <code>_POSIX_SEMAPHORES</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
588 589 590 591	<i>PCTS_sem_post</i>	<b>TRUE</b> if <code>_POSIX_SEMAPHORES</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
592 593 594 595	<i>PCTS_sem_trywait</i>	<b>TRUE</b> if <code>_POSIX_SEMAPHORES</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
596 597 598 599	<i>PCTS_sem_unlink</i>	<b>TRUE</b> if <code>_POSIX_SEMAPHORES</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
600 601 602 603	<i>PCTS_sem_wait</i>	<b>TRUE</b> if <code>_POSIX_SEMAPHORES</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
604 605 606 607	<i>PCTS_shm_open</i>	<b>TRUE</b> if <code>_POSIX_SHARED_MEMORY_OBJECTS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
608 609 610 611	<i>PCTS_shm_unlink</i>	<b>TRUE</b> if <code>_POSIX_SHARED_MEMORY_OBJECTS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
612 613 614 615	<i>PCTS_sigqueue</i>	<b>TRUE</b> if <code>_POSIX_REALTIME_SIGNALS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
616 617 618 619	<i>PCTS_sigtimedwait</i>	<b>TRUE</b> if <code>_POSIX_REALTIME_SIGNALS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .



	<b>Symbol</b>	<b>Value</b>
620 621 622 623	<i>PCTS_sigwaitinfo</i>	<b>TRUE</b> if <code>_POSIX_REALTIME_SIGNALS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
624 625 626	<i>PCTS_timer_create</i>	<b>TRUE</b> if <code>_POSIX_TIMERS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
627 628 629	<i>PCTS_timer_delete</i>	<b>TRUE</b> if <code>_POSIX_TIMERS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
630 631 632	<i>PCTS_timer_getoverrun</i>	<b>TRUE</b> if <code>_POSIX_TIMERS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
633 634 635	<i>PCTS_timer_gettime</i>	<b>TRUE</b> if <code>_POSIX_TIMERS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
636 637 638	<i>PCTS_timer_settime</i>	<b>TRUE</b> if <code>_POSIX_TIMERS</code> is defined or the implementation supports the function as described in POSIX.1b {3}, else <b>FALSE</b> .
639 640	<i>PCTS_write</i>	<b>TRUE</b> if the implementation supports the <code>write ()</code> , function. (Always <b>TRUE</b> ).

THIS PAGE IS INTENTIONALLY BLANK.

## Section 2: Terminology and General Requirements

### 180 2.1 Conventions

181 This standard uses the following typographic conventions:

182 (1) The *italic* font is used for:

- 183 -- Cross references to defined terms within 1.3, 2.2.1, and 2.2.2; symbolic parameters that are  
184 generally substituted with real values by the application
- 185 -- C language data types and function names (except in function Synopsis subclauses
- 186 -- Global external variable names
- 187 -- General Assertion and General Documentation Assertion references
- 188 -- The **bold** font is used with a word in all capital letters, such as **PATH** to represent an  
189 environmental variable. It is also used for the term “**NULL** pointer”
- 190 -- The `constant-width` (Courier) font is used:
  - 191 - For C language data types and function names within function Synopsis subclauses
  - 192 - To illustrate examples of system input or output where exact usage is depicted
  - 193 - For references to utility names and C language headers
  - 194 - Symbolic constants returned by many functions as error numbers are represented as:
    - 195 [ERRNO]
  - 196 - Symbolic Constants or limits defined in certain headers are represented as:
    - 197 [LIMIT]
  - 198 - Test method macros are represented as **M\_test\_method\_macro** ().

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

202 The conventions listed previously are for ease of reading only. Editorial inconsistencies in the use of typography<sup>6</sup>  
203 are unintentional and have no normative meaning in this standard.

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

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

## 208 **2.2 Definitions**

### 209 **2.2.1 Terminology**

210 There are no requirements for conforming implementation in this subclause.

### 211 **2.2.2 General Terms**

212 There are no requirements for conforming implementations in this subclause.

213 **2.2.2.1 absolute pathname:** there are no requirements for conforming implementation s in this subclause.

214 **2.2.2.2 access mode:** There are no requirements for conforming implementations in this subclause.

215 **2.2.2.3 address space:** There are no requirements for conforming implementations in this subclause.

216 **2.2.2.4 appropriate privileges:** There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no  
217 POSIX.1b {3} assertions.

218 **2.2.2.5 arm (a timer):** There are no requirements for conforming implementations in this subclause.

219 **2.2.2.6 asynchronous I/O operation:** There are no requirements for conforming implementations in this  
220 subclause.

221 **2.2.2.7 asynchronous I/O completion:** There are no requirements for conforming implementations in this  
222 subclause.

223 **2.2.2.8 background process:** There are no requirements for conforming implementations in this subclause.

224 **2.2.2.9 background process group:** There are no requirements for conforming implementations in this  
225 subclause.

226 **2.2.2.10 block special file:** There are no requirements for conforming implementations in this subclause.

227 **2.2.2.11 blocked process:** There are no requirements for conforming implementations in this subclause.

228 **2.2.2.12 character:** There are no requirements for conforming implementation s in this subclause.

229 **2.2.2.13 character special file:** There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no  
230 POSIX.1b {3} assertions.

231 **2.2.2.14 child process:** There are no requirements for conforming implementations in this subclause.

232 **2.2.2.15 clock:** There are no requirements for conforming implementations in this subclause.

233 **2.2.2.16 clock tick:** There are no requirements for conforming implementations in this subclause.

234 **2.2.2.17 controlling process:** There are no requirements for conforming implementations in this subclause.

235 **2.2.2.18 controlling terminal:** There are no requirements for conforming implementations in this subclause.

236 **2.2.2.19 current working directory:** There are no requirements for conforming implementations in this  
237 subclause.

238 **2.2.2.20 device:** There are no requirements for conforming implementations in this subclause.

- 239 **2.2.2.21 directory:** There are no requirements for conforming implementations in this subclause.
- 240 **2.2.2.22 directory entry [link]:** There are no requirements for conforming implementations in this subclause.
- 241 **2.2.2.23 direct I/O:** There are no requirements for conforming implementations in this subclause.
- 242 **2.2.2.24 disarm (a timer):** There are no requirements for conforming implementation of this subclause.
- 243 **2.2.2.25 drift rate (of a clock):** There are no requirements for conforming implementations in this subclause.
- 244 **2.2.2.26 dot:** There are no requirements for conforming implementations in this subclause.
- 245 **2.2.2.27 dot-dot:** There are no requirements for conforming implementations in this subclause.
- 246 **2.2.2.28 effective group ID:** There are no requirements for conforming implementations in this subclause.
- 247 **2.2.2.29 effective user ID:** There are no requirements for conforming implementations in this subclause.
- 248 **2.2.2.30 empty directory:** There are no requirements for conforming implementations in this subclause.
- 249 **2.2.2.31 empty string [null string]:** There are no requirements for conforming implementations in this subclause.
- 250 **2.2.2.32 Epoch:** There are no requirements for conforming implementations in this subclause.
- 251 **2.2.2.33 feature test macro:** There are no requirements for conforming implementations in this subclause.
- 252 **2.2.2.34 FIFO special file [FIFO]:** There are no requirements for conforming implementations in this subclause.
- 253 **2.2.2.35 file:** There are only IEEE std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.
- 254 **2.2.2.36 file description:** There are no requirements for conforming implementations in this subclause.
- 255 **2.2.2.37 file descriptor:** There are no requirements for conforming implementations in this subclause.
- 256 **2.2.2.38 file group class:** There are only IEEE std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3}
- 257 assertions.
- 258 **2.2.2.39 file mode:** There are no requirements for conforming implementations in this subclause.
- 259 **2.2.2.40 filename:**
- 260 **(IEEE Std 2003.1-1992 {4} GA02**
- 261 *UNUSED*
- 262 **M\_GA\_portableFilenames(*function*()) =**
- 263 **TEST:** The interface *function* () supports filenames containing any of the characters in the portable
- 264 filename character set.
- 265 **TR** Test for filenames containing the following characters:
- 266 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
- 267 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
- 268 0 1 2 3 4 5 6 7 8 9 . \_ -
- 269 The last three characters are the period, underscore, and hyphen characters, respectively.
- 270 **GA\_portableFilenames**
- 271 **FOR:** *execl()*, *execle()*, *execv()*, *execve()*, *exelp()*, *execvp()*, *opendir()*, *chdir()*, *open()*, *creat()*,
- 272 *link()*, *existing*, *link() new*, *mkdir()*, *mkfifo()*, *unlink()*, *rmdir()*, *rename()*, *old*, *rename()*,
- 273 *new*, *stat()*, *access()*, *chmod()*, *chown()*, *utime()*, *pathconf()*, *fopen()*, *freopen()*, *remove()*,
- 274 *tar* format creating utility, and *cpio* format creating utility.
- 275 **M\_GA\_portableFilenames(*function*())**
- 276 *Conformance for definition: PASS*

277 (IEEE Std 2003.1-1992 {4} GA03  
278 *UNUSED*

279 *M\_GA\_upperLowerNames(function()) =*

280 **TEST:** The interface *function()* differentiates between upper and lower case characters in  
281 filenames.

282 **GA\_upperLowerNames**

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

287 *M\_GA\_upperLowerNames(function())*

288 *Conformance for definitions: PASS*

289 **2.2.2.41 file offset:** There are no requirements for conforming implementations in this subclause.

290 **2.2.2.42 file other class:** There are no requirements for conforming implementations in this subclause.

291 **2.2.2.43 file owner class:** There are no requirements for conforming implementations in this subclause.

292 **2.2.2.44 file permission bits:** There are no requirements for conforming implementations in this subclause.

293 **2.2.2.45 file serial number:** There are no requirements for conforming implementations in this subclause.

294 **2.2.2.46 file system:** There are no requirements for conforming implementations in this subclause.

295 **2.2.2.47 first open (of a file):** *cpio* format creating utility. **There are no requirements for conforming**  
296 **implementations in this subclause.**

297 **2.2.2.48 foreground process:** There are no requirements for conforming implementations in this subclause.

298 **2.2.2.49 foreground process group:** There are no requirements for conforming implementations in this subclause.

299 **2.2.2.50 foreground process group ID:** There are no requirements for conforming implementations in this  
300 subclause.

301 **2.2.2.51 group ID:** There are no requirements for conforming implementations in this subclause.

302 **2.2.2.52 job control:** There are no requirements for conforming implementations in this subclause.

303 **2.2.2.53 last close (of a file)** There are no requirements for conforming implementations in this subclause.

304 **2.2.2.54 link:** There are no requirements for conforming implementations in this subclause.

305 **2.2.2.55 link count:** There are no requirements for conforming implementations in this subclause.

306 **2.2.2.56 login:** There are no requirements for conforming implementations in this subclause.

307 **2.2.2.57 login name:** There are no requirements for conforming implementations in this subclause.

308 **2.2.2.58 map:** There are no requirements for conforming implementations in this subclause:

309 **2.2.2.59 memory object:** There are no requirements for conforming implementations in this subclause.

310 **2.2.2.60 memory-resident:** There are no requirements for conforming implementation s in this subclause.

311 **2.2.2.61 message:** There are no requirements for conforming implementations in this subclause.

- 312 **2.2.2.62 message queue:** There are no requirements for conforming implementations in this subclause.
- 313 **2.2.2.63 mode:** There are no requirements for conforming implementations in this subclause.
- 314 **2.2.2.64 null string:** There are no requirements for conforming implementations in this subclause.
- 315 **2.2.2.65 open file:** There are no requirements for conforming implementations in this subclause.
- 316 **2.2.2.66 open file description:** There are no requirements for conforming implementations in this subclause.
- 317 **2.2.2.67 orphaned process group:** There are no requirements for conforming implementations in this subclause.
- 318 **2.2.2.68 page:** There are no requirements for conforming implementations in this subclause.
- 319 **2.2.2.69 parent directory:** There are no requirements for conforming implementations in this subclause.
- 320 **2.2.2.70 parent process:** There are no requirements for conforming implementations in this subclause.
- 321 **2.2.2.71 parent process ID:** There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3}  
322 assertions.
- 323 **2.2.2.72 path prefix:** There are no requirements for conforming implementations in this subclause.
- 324 **2.2.2.73 pathname:** There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3}  
325 assertions.
- 326 **2.2.2.74 pathname component:** There are no requirements for conforming implementations in this subclause.
- 327 **2.2.2.75 persistence:** There are no requirements for conforming implementations in this subclause.
- 328 **2.2.2.76 pipe:** There are no requirements for conforming implementations in this subclause.
- 329 **2.2.2.77 portable filename character set:** There are no requirements for conforming implementations in this  
330 subclause.
- 331 **2.2.2.78 preallocation:** There are no requirements for conforming implementations in this subclause.
- 332 **2.2.2.79 preempted process:** There are no requirements for conforming implementations in this subclause.
- 333 **2.2.2.80 priority:** There are no requirements for conforming implementations in this subclause.
- 334 **2.2.2.81 priority-based scheduling:** There are no requirements for conforming implementations in this subclause.
- 335 **2.2.2.82 privilege:** There are no requirements for conforming implementations in this subclause.
- 336 **2.2.2.83 process:** There are no requirements for conforming implementations in this subclause.
- 337 **2.2.2.84 process group:** There are no requirements for conforming implementations in this subclause.
- 338 **2.2.2.85 process group ID:** There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3}  
339 assertions.
- 340 **2.2.2.86 process group leader:** There are no requirements for conforming implementations in this subclause.
- 341 **2.2.2.87 process group lifetime:** There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b  
342 {3} assertions.
- 343 **2.2.2.88 process ID:** There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3}  
344 assertions.

345 **2.2.289 process lifetime:** There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3}  
 346 assertions.

347 **2.2.2.90 process list:** There are no requirements for conforming implementations in this subclause.

348 **2.2.2.91 read-only file system:** There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b  
 349 {3} assertions.

350 **2.2.2.92 real group ID:** There are no requirements for conforming implementations in this subclause.

351 **2.2.2.93 real user ID:** There are no requirements for conforming implementations in this subclause.

352 **2.2.2.94 referenced shared memory object:** There are no requirements for conforming implementations in this  
 353 subclause.

354 **2.2.2.95 regions:** There are no requirements for conforming implementations in this subclause.

355 **2.2.2.96 regular file:** There are no requirements for conforming implementations in this subclause.

356 **2.2.2.97 relative pathname:** There are no requirements for conforming implementations in this subclause.

357 **2.2.2.98 (time) resolution:** There are no requirements for conforming implementations in this subclause.

358 **2.2.2.99 root directory:** There are no requirements for conforming implementations in this subclause.

359 **2.2.2.100 runnable process:** There are no requirements for conforming implementations in this subclause.

360 **2.2.2.101 running process:** There are no requirements for conforming implementations in this subclause.

361 **2.2.2.102 saved set-group-ID:** There are no requirements for conforming implementations in this subclause.

362 **2.2.2.103 saved set-user-ID:** There are no requirements for conforming implementations in this subclause.

363 **2.2.2.104 scheduling:** There are no requirements for conforming implementations in this subclause.

364 **2.2.2.105 scheduling policy:**

365 **D\_1 TEST:** The PCD.1b shall define in subclause 2.2.2.105 the manner in which each of the scheduling  
 366 policies may modify the priorities or otherwise affect the ordering of processes at each of the  
 367 following occurrences

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

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

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

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

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

374 *Conformance for definitions: PASS*

375 **D\_2 TEST:** The PCD.1b defines in subclause 2.2.2.105 under what other circumstances and in what manner  
 376 each scheduling policy may modify the priorities affect the ordering of processes.

377 *Conformance for definitions: PASS*

378 **2.2.2.106 seconds since the Epoch:** There are no requirements for conforming implementations in this subclause.



379 **2.2.2.107 semaphore:** There are no requirements for conforming implementations in this subclause.

380 **2.2.2.108 semaphore lock operation:**

381 **R\_1 FOR:** *sem\_init()* and *sem\_open()*  
 382 **IF** *PCTS\_sem\_wait* **THEN**  
 383 **IF** *PCTS\_function* **THEN**  
 384 **SETUP:** Create a semaphore using *function* ().  
 385 **TEST:** When a call to *sem\_wait* (*sem*) complete successfully, the interface returns a value  
 386 of 0 and the semaphore designated by *sem* is locked by the semaphore lock  
 387 operation.  
 388 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag *PCTS\_GAP\_sem\_init*;  
 389 that is, generate a *NO\_TEST\_SUPPORT* test result code if there is not a way to get  
 390 appropriate privilege to call *sem\_init()*.  
 391 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
 392 assertion is to be read by substituting *function()* with the current function specified  
 393 in the FOR clause. The name of the function also is to be substituted for each  
 394 occurrence in the construct *PCTS\_function*  
 395 **ELSE** *NO\_TEST\_SUPPORT*  
 396 **ELSE** *NO\_OPTION*  
 397 **SEE:** Assertion *sem\_wait* in §11.2.7.2

398 **2.2.2.109 semaphore unlock operation:**

399 **R\_2 FOR:** *sem\_init()* and *sem\_open()*  
 400 **IF** *PCTS\_sem\_post* **THEN**  
 401 **IF** *PCTS\_function* **THEN**  
 402 **SETUP:** Create a semaphore using *function* ().  
 403 **TEST:** A successful call to *sem\_post()* unlocks the semaphore referenced by *sem* by  
 404 performing the semaphore unlock operation on that semaphore, and returns the value  
 405 zero.  
 406 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag *PCTS\_GAP\_sem\_init*;  
 407 that is, generate a *NO\_TEST\_SUPPORT* test result code if there is not a way to get  
 408 appropriate privilege to call *sem\_init()*.  
 409 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
 410 assertion is to be read by substituting *function()* with the current function specified  
 411 in the FOR clause. The name of the function also is to be substituted for each  
 412 occurrence in the construct *PCTS\_function*  
 413 **ELSE** *NO\_TEST\_SUPPORT*  
 414 **ELSE** *NO\_OPTION*  
 415 **SEE:** Assertion *sem\_post* in §11.2.7.2

416 **2.2.2.110 session:** There are no requirements for conforming implementations in this subclause.

417 **2.2.2.111 session leader:** There are no requirements for conforming implementations in this subclause.

418 **2.2.2.112 session lifetime:** There are no requirements for conforming implementations in this subclause.

419 **2.2.2.113 shared memory object:** There are no requirements for conforming implementations in this subclause.

420 **2.2.2.114 signal:** There are no requirements for conforming implementations in this subclause.

421 **2.2.2.115 slash:** There are no requirements for conforming implementations in this subclause.

422 **2.2.2.116 supplementary group ID:** There are only IDDD Std 2003.1-1992 {4} assertions in this subclause; no  
 423 POSIX.1b {3} assertions.

424 **2.2.2.117 successfully transferred:** There are no requirements for conforming implementations in this subclause.

425 **2.2.2.118 synchronized I/O completion:** There are no requirements for conforming implementations in this  
 426 subclause.

427 **2.2.2.119 synchronized I/O data integrity completion:**

428 **GA\_syncIODataIntegrityRead**

429 **FOR:** *read()*, *aio\_read()*, and *lio\_listio()*

430 **IF** *PCTS\_function* and {POSIX\_SYNCH\_IO} **THEN**

431 **SETUP:** Open a file by calling *open()* with O\_RSYNC and O\_DSYNC set in the *oflag* parameter.

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

435 **TR:** Test for regular files.

436 **ELSE NO\_OPTION**

437 *Conformance for definitions: PASS, NO\_TEST, NO\_OPTION*

438 **GA\_syncIODataIntegrityWbeforeR**

439 **FOR:** *write()*, *aio\_write()*, and *lio\_listio()*

440 **IF** *PCTS\_function* and {POSIX\_SYNCH\_IO} **THEN**

441 **SETUP:** Open a file by calling *open()* with O\_DSYNC set in the *oflag* parameter.

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

445 **TR:** Test for regular files and, if PCTS\_GTI\_DEVICE, terminals.

446 **ELSE NO\_OPTION**

447 *Conformance for definitions: PASS, NO\_TEST, NO\_OPTION*

448 **GA\_syncIODataIntegrityWrite**

449 **FOR:** *write()*, *aio\_write()*, and *lio\_listio()*

450 **IF** *PCTS\_funtion* and {POSIX\_SYNCH\_IO} **THEN**

451 **SETUP:** Open a file by calling *open()* with O\_DSYNC set in the *oflag* parameter.

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

455 **TR:** Test for regular files and, if PCTS\_GTI\_DEVICE, terminals.

456 **ELSE NO\_OPTION**

457 *Conformance for definitions: PASS, NO\_TEST, NO\_OPTION*

458 **2.2.2.120 synchronized I/O file integrity completion:**

459 **GA\_syncIOFileIntegrityRead**

460 **FOR:** *read()*, *aio\_read()*, and *lio\_listio()*

461 **IF** *PCTS\_function* and {POSIX\_SYNCH\_IO} **THEN**

462 **SETUP** Open a file by calling *open()* with O\_RSYNC and O\_SYNC set in the *oflag*  
 463 parameter.

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

- 469
- 470 6. File mode.
  - 471 2. File serial number
  - 472 3. ID of device containing this file.
  - 473 4. Number of links.
  - 474 5. User ID of the owner of the file.

- 475                   6.    Group ID of the group of the file.  
 476  
 477                   7.    The file size in bytes.  
 478                   8.    Time of last access.  
 479                   9.    Time of last data modification.  
 480                   10.   Time of last file status change.

481                   **TR:**    Test for regular files.  
 482                   **ELSE NO\_OPTION**  
 483                   *Conformance for definitions: PASS, NO\_TEST, NO\_OPTION*

484   **GA\_syncIOFileIntegrityWrite**

485                   **FOR:**    write(), aio\_write(), and lio\_listio()  
 486                   **IF** *PCTS\_function* and {POSIX\_SYNCH\_IO} **THEN**  
 487                   **TEST:**    At the time that the synchronized write operation initiated by calling *function()*  
 488                                   occurs, the data are written to the physical medium containing the file and the  
 489                                   following file attributes are also written to the physical medium containing the file  
 490                                   prior to returning to the calling process:

- 491                   1.    File mode.  
 492                   2.    File serial number.  
 493                   3.    ID of device containing this file.  
 494                   4.    Number of links.  
 495                   5.    User ID of the owner of the file.  
 496                   6.    Group ID of the group of the file.  
 497                   7.    The file size in bytes.  
 498                   8.    Time of last access.  
 499                   9.    Time of last data modification.  
 500                   10.   Time of last file status change.

501                   **TR:**    Test for regular files.  
 502                   **ELSE NO\_OPTION**  
 503                   *Conformance for definitions: PASS, NO\_TEST, NO\_OPTION*

504   **2.2.2.121 synchronized I/O operation:** There are no requirements for conforming implementations in this  
 505   subclause.

506   **2.2.2.122 synchronous I/O operation:** There are no requirements for conforming implementations in this  
 507   subclause.

508   **2.2.2.123 system:** There are no requirements for conforming implementations in this subclause.

509   **2.2.2.124 system crash:** There are no requirements for conforming implementations in this subclause.

510   **2.2.2.125 system process:** There are no requirements for conforming implementations in this subclause.

511   **2.2.2.126 system reboot:**

512 **D\_3 TEST:** The PCD.1b defines in subclause 2.2.2.126 the implementations defined sequence of events  
 513 (called a system reboot) that may result in the loss of transitory data, i.e., data that is not saved  
 514 in permanent storage.  
 515 *Conformance for definitions: PASS*

516 **2.2.2.127 terminal [terminal device]:** There are no requirements for conforming implementations in this  
 517 subclause.

518 **2.2.2.128 timer:** There are no requirements for conforming implementations in this subclause.

519 **2.2.2.129 timer overrun:** There are no requirements for conforming implementations in this subclause.

520 **2.2.2.130 user ID:** There are no requirements for conforming implementations in this subclause.

521 **2.2.2.131 user name:** There are no requirements for conforming implementations in this subclause.

522 **2.2.2.132 working directory [current working directory]:** There are no requirements for conforming  
 523 implementations in this subclause.

## 524 2.2.3 Abbreviations

525 There are no requirements for conforming implementations in this subclause.

526 See subclause 1.4.2 for abbreviations related to this standard.

## 527 2.3 General Concepts

528 **2.3.1 extended security controls:** There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no  
 529 POSIX.1b {3} assertions.

530 **2.3.2 file access permissions:**  
 531 (IEEE Std 2003.1-1992 {4} GA04  
 532 *UNUSED*

533 ***M\_GA\_AP\_overrideFileAccess(function()) =***  
 534 **IF** the IUT provides a mechanism for creating processes with the appropriate privilege to override a  
 535 file access control mechanism **THEN**  
 536 **SETUP:** The process has appropriate privileges for file access.  
 537 **TEST:** A call to the interface *function()* that needs read, write, or search access to the  
 538 *pathname* argument is granted access to the file when access would otherwise be  
 539 denied..  
 540 **ELSE NO\_OPTION**

541 **GA\_AP\_overrideFileAccess**  
 542 **FOR:** *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execlp(),*  
 543 *execvp(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(),*  
 544 *rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*  
 545 ***M\_GA\_AP\_overrideFileAccess(function())***  
 546 *Conformance for General Concepts; PASS, NO\_TEST, NO\_OPTION*

547 (IEEE Std 2003.1-1992 {4} GA05  
 548 *UNUSED*

549 ***M\_GA\_AP\_overrideExecAccess(function()) =***  
 550 **IF** the IUT provides a mechanism for creating processes with the appropriate privilege to override a  
 551 file access control mechanism **THEN**  
 552 **SETUP:** The process has appropriate privilege for the file access and execute permission is  
 553 granted to at least one user of the file.

554                   **TEST:**    A call to the interface *function()* that needs execute permission to the *path* or *file*  
555    argument is granted execute access to the file when access would otherwise be  
556    denied.  
557                   **ELSE NO\_OPTION**

558 **GA\_AP\_overrideExecAccess**  
559                   **FOR:**    *execl(), execl(), execv(), execve(), execlp(), and execvp().*  
560                   **M\_GA\_AP\_overrideExecAccess(function())**  
561                   *Conformance for General Concepts: PASS, NO\_TEST, NO\_OPTION*

562 **(IEEE Std 2003.1-1992 {4} GA06**  
563                   **UNUSED**

564 **M\_GA\_AP\_classAccess(function()) =**  
565                   **IF** the IUT provides a mechanism for creating processes with the appropriate privilege to override a  
566                   file access control mechanism **THEN**  
567                    **SETUP:**    The process does not have appropriate privilege to override the file access control  
568    mechanism and the process requires read, write, execute, or search access to the  
569    *pathname* or *file* argument of the interface *function()*.  
570                    **TEST:**    A call to the interface *function()* is granted access to the file when the required access  
571    permission bit is set for the class (file owner class, file group class, or file other  
572    class) to which the process belongs.  
573                   **ELSE NO\_OPTION**

574 **GA\_AP\_classAccess**  
575                   **FOR:**    *access(), chdir(), chmod(), chown(), creat(), execl(), execl(), execv(), execve(), execlp(),*  
576    *execvp(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(),*  
577    *rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*  
578                   **M\_GA\_AP\_ClassAccess(function())**  
579                   *Conformance for General Concepts: PASS, NO\_TEST, NO\_OPTION*

580 **(IEEE Std 2003.1-1992 {4} GA07**  
581                   **UNUSED**

582 **M\_GA\_AdditionalAccessControl() =**  
583                   **IF** IUT provides additional file access control mechanisms **THEN**  
584                    **TEST:**    Any additional file access control mechanism shall only further restrict the access  
585    permissions defined by the file permission bits.  
586                   **ELSE NO\_OPTION**

587 **GA\_AdditionalAccessControl**  
588                   **MP\_GA\_AdditionalAccessControl()**  
589                   *Conformance for General Concepts: PASS, NO\_TEST, NO\_OPTION*

590 **(IEEE Std 2003.1-1992 {4} GA08**  
591                   **UNUSED**

592 **MP\_GA\_AlternateAccessControl() =**  
593                   **IF** IUT provides alternate file access control mechanisms **THEN**  
594                    **TEST:**    Any alternate file access control mechanism specifies file permission bits for the file  
595    owner class, file group class, and file other class of the file corresponding to the  
596    access permissions to be returned by *stat()* and *fstat()*.  
597                   **ELSE NO\_OPTION**

598 **GA\_AlternateAccessControl**  
599                   **M\_GA\_AlternateAccessControl()**  
600                   *Conformance for General Concepts, PASS, NO\_TEST, NO\_OPTION*

601 **(IEEE Std 2003.1-1992 {4} GA09**  
602                   **UNUSED**

603 (IEEE Std 2003.1-1992 {4}) GA10  
604 *UNUSED*

605 *M\_GA\_AltAccessEnable* () =  
606 **IF** IUT provides alternate file access control mechanisms **THEN**  
607 **TEST:** The alternate file access control mechanisms can be enabled only by explicit user action,  
608 on a per-file basis by the file owner or a user with the appropriate privilege.  
609 **ELSE** *NO\_OPTION*

610 **GA\_AltAccessEnable**  
611 *M\_GA\_AltAccessEnable*()  
612 *Conformance for General Concepts: PASS, NO\_TEST, NO\_OPTION*

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

615 *M\_GA\_AltAccessDisable*() =  
616 **IF** IUT provides alternate file access control mechanisms **THEN**  
617 **TEST:** The alternate file access control mechanisms will be disabled for a file after the file  
618 permission bits are changed for that file with *chmod*().  
619 **ELSE** *NO\_OPTION*

620 **GA\_AltAccessDisable**  
621 *M\_GA\_AltAccessDisable*()  
622 *Conformance for General Concepts: PASS, NO\_TEST, NO\_OPTION*

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

625 **2.3.3. file hierarchy:** There are no requirements for conforming implementations in this subclause.

626 **2.3.4 filename portability:** There are no requirements for conforming implementations in this subclause.

627 **2.3.5 file times update:**  
628 (IEEE Std 2003.1-1992 {4})R01  
629 *UNUSED*

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

632 *M\_GA\_StatTimeUpdate* (*function*()) =  
633 **TEST:** The interface *function*() when called updates all time-related fields marked for update and  
634 does not update any time-related fields not marked for update.

635 **GA\_StatTimeUpdate**  
636 **FOR:** *stat*() and *fstat*().  
637 *M\_GA\_StatTimeUpdate*(*function*())  
638 *Conformance for General Concepts: PASS*

639 *M\_GA\_NoOpenTimeUpdate*(*function*()) =  
640 **TEST:** All fields that are marked for update are updated when the file is no longer open by any  
641 process.

642 **GA\_NoOpenTimeUpdate**  
643 **FOR:** *close*() and *fclose*().  
644 *M\_GA\_NoOpenTimeUpdate* (*function*())  
645 *Conformance for General Concepts: PASS*

646 NOTE: This assertion is missing in 2003.1

647 **(IEEE Std 2003.1-1992 {4}\_GA13**  
 648 *UNUSED*

649 ***M\_GA\_NoROFSTimeUpdate(function()) =***  
 650 **TEST:** Time-related field updates are not done for files on read-only file systems.

651 **GA\_NoROFSTimeUpdate**  
 652 **FOR:** *acc(), chmod(), chown(), creat(), link() existing, link() new, mkdir(), mkfifo(), open(),*  
 653 *rename() new, rename() old, rmdir(), unlink(), and utime().*  
 654 ***M\_GA\_noROFSTimeUpdate(function())***  
 655 *Conformance for General Concepts: PASS*

### 656 **2.3.6 pathname resolution:**

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

661 **(IEEE Std 2003.1-1992 {4}) GA14**

662 *UNUSED*

663 ***M\_GA\_PRDot(function()) =***  
 664 **TEST:** A call to the interface *function()* with a *path* or *file* argument where the first filename  
 665 component is "." and the argument does not begin with a "/" (slash resolves the *path* or *file*  
 666 argument by locating the second filename component (when specified) in the current  
 667 working directory

668 **GA\_PRDot**  
 669 **FOR:** *access(), chdir(), chmod(), chmod(), chown(), creat(), execl(), execl(), execv(), execve(),*  
 670 *execlp(), execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(),*  
 671 *mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink() and*  
 672 *utime().*  
 673 ***M\_GA\_PRDot(function())***  
 674 *Conformance for General Concepts: PASS*

675 **(IEEE Std 2003.1-1992 {4}\_GA15**

676 *UNUSED*

677 ***M\_GA\_PRSlash(function()) =***  
 678 **TEST:** A call to the interface *function()* with a *path* or *file* argument pointing to the string "/"  
 679 resolves the *path* or *file* argument to the root directory of the process.

680 **GA\_PRSlash**  
 681 **FOR:** *access(), chdir(), chmod(), chown(), creat(), execl(), execl(), execv(), execve(), execp(),*  
 682 *execpt(), fopen(), freopen(), open(), opendir(), oink() existing, link() new, mkdir(), mkfifo(),*  
 683 *pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*  
 684 ***M\_GA\_PRSlash(function())***  
 685 *Conformance for General Concepts: pass*

686 **(IEEE Std 2003.1-1992 {4}) GA16**

687 *UNUSED*

688 ***M\_GA\_PR3Slash(function()) =***  
 689 **TEST:** A call to the interface *function()* with a *path* or *file* argument pointing to the string "/"  
 690 resolves the *path* or *file* argument to the root directory of the process.

691 **GA\_PR3Slash**

692           **FOR:**     *access(), chdir(), chmod(), chown(), creat(), execl(), execl(), execv(), execve(), execlp(),*  
 693                    *execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(),*  
 694                    *pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*  
 695            **M\_GA\_PR3Slash***function()*  
 696            Conformance for General Concepts: PASS

697    **(IEEE Std 2003.1-1992 {4}) GA17**  
 698            **UNUSED**

699    **M\_GA\_PRSlashesPath***function()* =  
 700            **TEST:**    A call to the interface *function()* with a *path* or *file* argument pointing to a string that starts  
 701                    with either a single slash ( "/" ) or three or more slashes resolves the *path* or *file* argument  
 702                    by locating the first filename component of the argument in the root directory of the  
 703                    process.

704    **GA\_PRSlashesPath**  
 705            **FOR:**     *access(), chdir(), chmod(), chown(), creat(), execl(), execl(), execv(), execve(), execl(),*  
 706                    *execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(),*  
 707                    *pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*  
 708            **M\_GA\_PRSlashesPath***function()*  
 709            Conformance for General Concepts: PASS

710    **(IEEE Std 2003.1-1992 {4}) GA18**  
 711            **UNUSED**

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

718    **GA\_PRDotDot**  
 719            **FOR:**     *access(), chdir(), chmod(), chown(), creat(), execl(), execl(), execv(), execve(), execl(),*  
 720                    *execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(),*  
 721                    *pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*  
 722            **M\_GA\_PRDotDot***function()*  
 723            Conformance for General Concepts; PASS

724    **(IEEE Std 2003.1-1992 {4}) GA19**  
 725            **UNUSED**

726    **M\_GA\_PRRelativeSlash***function()* =  
 727            **TEST:**    A call to the interface *function()* ( ) with a *path* or *file* argument pointing to the string "F1/"  
 728                    and "F1" is a directory resolves the *path* or *file* argument by locating F1 "F1" in the current  
 729                    working directory.

730    **GA\_PRRelativeSlash**  
 731            **FOR:**     *access(), chdir(), chmod(), chown(), creat(), execl(), execl(), execv(), execve(), execl(),*  
 732                    *execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(),*  
 733                    *pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*  
 734            **M\_GA\_PRRelativeSlash***function()*  
 735            Conformance for General Concepts: PASS

736    **(IEEE Std 2003.1-1992 {4}) GA20**  
 737            **UNUSED**

738    **M\_GA\_PRRelativeSlashSlash***function()* =



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

742    **GA\_PRRelativeSlashSlash**

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

746            **M\_GA\_PRRelativeSlashSlash**(*function()*)  
747            Conformance for General Concepts: PASS

748    **M\_GA\_PRRenameRelativeSlashSlash**(*function()*) =

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

752    **GA\_PRRenameRelativeSlashSlash**

753            **FOR:**       *rename()new*  
754            **M\_GA\_PRRenameRelativeSlashSlash**(*function()*)  
755            Conformance for general Concepts: PASS

756    **(IEEE Std 2003.1-1992 {4})GA21**

757            UNUSED

758    **M\_GA\_PRRelativeCWD**(*function()*) =

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

762    **GA\_PRRelativeCWD**

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

766            **M\_GA\_PRRelativeCWD**(*function()*)  
767            Conformance for General Concepts: PASS

768    **(IEEE Std 2003.1-1992 {4}\_ GA22**

769            UNUSED

770    **M\_GA\_PRRelativeDotCWD**(*function()*) =

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

774    **GA\_PRRelativeDotCWD**

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

778            **M\_GA\_PRRelativeDotCWD**(*function()*)  
779            Conformance for General Concepts: PASS

780    **(IEEE Std 2003.1-1992 {4} GA23**

781            UNUSED

782    **M\_GA\_PRRelativeDotDotCWD**(*function()*) =

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

786    **GA\_PRRelativeDotDotCWD**

787           **FOR:**     *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execl(),*  
 788                    *execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(),*  
 789                    *pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*  
 790            **M\_GA\_PRRelativeDotDotCWD(function())**  
 791            *Conformance for General Concepts: PASS*

792   **(IEEE Std 2003.1-1992 {4}) GA24**  
 793            *UNUSED*

794   **M\_GA\_PRRelativeSlashSlashCWD(function()) =**  
 795            **TEST:**    A call to the interface *function()* () with a path or file argument pointing to the string  
 796                    "F1//F2" resolves the path or file argument by locating "F2" in the directory "F1" in the  
 797                    current working directory.

798   **GA\_PRRelativeSlashSlashCWD**  
 799            **FOR:**     *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execl(),*  
 800                    *execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(),*  
 801                    *pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*  
 802            **M\_GA\_PRRelativeSlashSlashCWD(function())**  
 803            *Conformance for General Concepts: PASS*

804   **(IEEE Std 2003.1-1992 {4}) GA25**  
 805            *UNUSED*

806   **M\_GA\_PPRnoTrunc(function()) =**  
 807            **IF** {POSIX\_NO\_TRUNC} is not supported in the specified directory **THEN**  
 808                    **TEST:**    A call to the interface *function()* () with a path or file argument that has a pathname  
 809                    component of more than {NAME\_MAX} bytes in a directory for which  
 810                    {POSIX\_NO\_TRUNC} is not supported resolves the pathname component by  
 811                    truncating it to {NAME\_MAX} bytes.  
 812            **ELSE NO\_OPTION**

813   **GA\_PPRnoTrunc**  
 814            **FOR:**     *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execl(),*  
 815                    *execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(),*  
 816                    *pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*  
 817            **M\_GA\_PPRnoTrunc(function())**  
 818            *Conformance for General Concepts: PASS, NO\_OPTION*

819   **M\_GA\_PPRnoTruncError(function()) =**  
 820            **IF** {POSIX\_NO\_TRUNC} is supported in the specified directory **THEN**  
 821                    **TEST:**    A call to the interface *function()* () with a path or file argument that has a pathname  
 822                    component of more than {NAME\_MAX} bytes in a directory for which  
 823                    {POSIX\_NO\_TRUNC} is supported generates an [ENAMETOOLONG] error.  
 824            **ELSE NO\_OPTION**

825   **GA\_PPRnoTruncError**  
 826            **FOR:**     *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execl(),*  
 827                    *execvp(), fopen(), freopen(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(),*  
 828                    *pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(), and utime().*  
 829            **M\_GA\_PPRnoTruncError(function())**  
 830            *Conformance for General Concepts: PASS, NO\_OPTION*

## 831   **2.4 Error Numbers**

832   **(IEEE Std 2003.1-1992 P{4}) 02**  
 833            *UNUSED*

834   **2        SETUP:**    Include the header `<errno.h>`.

835           **TEST:**    The error numbers[E2BIG], [EACCESS], [EAGAIN], [EBADF], EBADMSG, EBUSY],  
 836                    ]ECANCELED], [ECHILD], [EDEADLK], [EDOM], [EEXIST], [EFAULT], [EFBIG],  
 837                    [EINPROGRESS], [EINTR], [EINTR], [EINVAL], [EIO], [EISDIR], [EMFILE], [EMLINK],  
 838                    [EMSGSIZE], [ENAMETOOLONG], [ENFILE], [ENODEV], [ENOENT], [ENOEXEC],  
 839                    [ENOLCK], [ENOMEM], [ENOSPC], [ENOSYS], [ENOTDIR], [ENOTEMPTY],  
 840                    [ENOTTY], [ENXIO], [EPERM], [EPIPE], [ERANGE], [EROFS], [ESPIPE], [ESRCH], and  
 841                    [EXDEV] are defined, are nonzero, are distinct from each other, and can be represented in  
 842                    *errno*  
 843                    Conformance for Error numbers: PASS

844    (IEEE Std 2003.1-1992 {4}) DGA02  
 845                    *UNUSED*

846    ***M\_GD\_OptionalErrors*(function()) =**  
 847                    **IF** the IUT supports the detection of an optional error condition **THEN**  
 848                    **TEST:**    The PCD.1b contains the details of the optional error conditions detected in the  
 849                    subclause of the PCD.1b where the error values of the interface *function()* are  
 850                    described.  
 851                    **ELSE NO\_OPTION**

852    **GD\_OptionalErrors**  
 853                    **FOR:**    *access(), chown(), chosedir(), execl(), execl(), execv(), execve(), execlp(), execvp(), fcntl(),*  
 854                    *fork(), fpathconf(), getcwd(), opendir(), pathconf(), readdir(), sigaddset(), sigdelset(), and*  
 855                    *(sigismember()).*  
 856                    ***M\_GD\_OptionalErrors*(function())**  
 857                    Conformance for Error Numbers: PASS, NO\_OPTION

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

860    ***M\_GA\_OptionalErrorsUndetected*(function()) =**  
 861                    **IF** the IUT does not supports the detection of an optional error condition **THEN**  
 862                    **TEST:**    The action specified by a call to the interface *function()* that would otherwise generate  
 863                    an optional error condition will succeed.  
 864                    **ELSE NO\_OPTION**

865    **GA\_OptionalErrorsUndetected**  
 866                    **FOR:**    *access(), chown(), chosedir(), execl(), execl(), execv(), execve(), execlp(), execvp(), fcntl(),*  
 867                    *fork(), fpathconf(), getcwd(), opendir(), pathconf(), readdir(), sigaddset(), sigdelset(), and*  
 868                    *(sigismember()).*  
 869                    ***M\_GA\_OptionalErrorsUndetected*(function())**  
 870                    Conformance for Error Numbers: PASS, NO\_OPTION

## 871    2.5 Primitive System Data Types

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

## 873    2.6 Environment Description

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

876    ***M\_GA\_ExecNoSlash* () =**  
 877                    **TEST:**    The interfaces *execlp()* and *execvp()* use the path prefixes in the **PATH** environment  
 878                    variable only when their *file* argument does not contain a slash.

879    **GA\_ExecNoSlash**  
 880                    **FOR:**    *execlp()* and *execvp()*.  
 881                    ***M\_GA\_ExecNoSlash*()**  
 882                    Conformance for Environment Description: PASS

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

885 *M\_GA\_ExecColon()* =  
886 **TEST:** The search path used by the interface *execlp()* and *execvp()* uses the path prefixes in the  
887 **PATH** environment variable that are separated by a colon.

888 **GA\_ExecColon**  
889 **FOR:** *execlp()* and *execvp()*.  
890 *M\_GA\_ExecColon()*  
891 *Conformance for Environment Description: PASS*

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

894 *M\_GA\_ExecInsertSlash()* =  
895 **TEST:** The interfaces *execlp()* and *execvp()* insert a “/” between a nonzero-length path prefix in the  
896 **PATH** environment variable and the filename in the file argument when searching for an  
897 executable file.

898 **GA\_ExecInsertSlash**  
899 **FOR:** *execlp()* and *execvp()*.  
900 *M\_GA\_ExecInsertSlash()*  
901 *Conformance for environment Description: PASS*

902 (IEEE Std 2003.1-1992 {4}) GA30  
903 *UNUSED*

904 *M\_GA\_execTwoColons()* =  
905 **TEST:** The search path used by the interfaces *execlp()* and *execvp()* uses the current working  
906 directory as the path prefix corresponding to two adjacent colons, “::”, in the **PATH**  
907 environment variable.

908 **GA\_ExecTwoColons**  
909 **FOR:** *execlp()* and *execvp()*.  
910 *M\_GA\_ExecTwoColons()*  
911 *Conformance for Environment Description: PASS*

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

914 *M\_GA\_ExecInitialColon()* =  
915 **TEST:** The search path used by the interfaces *execlp()* and *execvp()* uses the current working  
916 directory as the path prefix when the value of **PATH** environment variable starts with in a  
917 “:”

918 **GA\_ExecInitialColon**  
919 **FOR:** *execlp()* and *execvp()*.  
920 *M\_GA\_ExecInitialColon()*  
921 *Conformance for Environment Description: PASS*

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

924 *M\_GA\_ExecTrailingColon()* =  
925 **TEST:** The search path used by the interfaces *execlp()* and *execvp()* uses the current working  
926 directory as the path prefix when the value of the **PATH** environment variable ends with  
927 in a “:”

- 928 **GA\_ExecTrailingColon**  
 929 **FOR:** *execlp()* and *execvp()*.  
 930 **M\_GA\_ExecTrailingColon()**  
 931 *Conformance for Environment Description: PASS*
- 932 **(IEEE Std 2003.1-1992 {4}) GA33**  
 933 *UNUSED*
- 934 **M\_GA\_ExecPathSearchOrder() =**  
 935 **TEST:** The interfaces *execlp()* and *execvp()* search the path prefixes in the **PATH** environment  
 936 variable from the beginning to the end until an executable program by the specified name  
 937 is found.
- 938 **GA\_ExecPathSearchOrder**  
 939 **FOR:** *execlp()* and *execvp()*  
 940 **M\_GA\_ExecPathSearchOrder()**  
 941 *Conformance for Environment Description: PASS*
- 942 **(IEEE Std 2003.1-1992 {4}) GA34**  
 943 *UNUSED*
- 944 **M\_GA\_EnvironCaseSensitive(function) =**  
 945 **TEST:** The interface function retains the unique identities of upper-and lowercase letters in the  
 946 environment and does not fold them together.
- 947 **GA\_EnvironCaseSensitive**  
 948 **FOR:** *execl(), execl(), execl(), execlp(), execv(), execve(), execlp(), execvp()* and *getenv()*.  
 949 **M\_GA\_EnvironCaseSensitive(function())**  
 950 *Conformance for Environment Description: PASS*
- 951 **(IEEE Std 2003.1-1992 {4}) GA35**  
 952 *UNUSED*
- 953 **M\_GA\_EnvironPortNames() =**  
 954 **TEST:** The interface supports environment variable names consisting of characters in the portable  
 955 filename character set.
- 956 **GA\_EnvironPortNames**  
 957 **FOR:** *execl(), execl(), execl(), execlp(), execv(), execve(), execlp(), execvp()* and *getenv()*.  
 958 **M\_GA\_EnvironPortNames(function())**  
 959 *Conformance for Environment Description: PASS*

## 960 **2.7 C Language Definitions**

### 961 **2.7.1 Symbols From the C Standard**

#### 962 **(IEEE Std 2003.1-1992 {4}) 04**

963 *UNUSED*

- 964 **4** **TEST:** Each of the headers *<aio.h>*, *<dirent.h>*, *<fcntl.h>*, *<grp.h>*,  
 965 *<limits.h>*, *<locale.h>*, *<mqueue.h>*, *<pwd.h>*, *<sched.h>*,  
 966 *<semaphore.h>*, *<signal.h>*, *<sys/mman.h>*, *<sys/stat.h>*,  
 967 *<sys/times.h>*, *<sys/wait.h>*, *<termios.h>*, *<time.h>*,  
 968 *<unistd.h>* and *<utime.h>* can be included more than once, in any  
 969 combination, in any order, and a symbol may be defined in more than one header with the  
 970 same value.  
 971 **NOTE:** The C Standard {2} headers that do not have additional requirements placed on them by  
 972 IEEE Std 1003.1b-1993 are not included because their testing should be done when  
 973 measuring conformance to the C Standard {2}.  
 974 *Conformance for C Language Definitions: PASS*

- 975 **4.1 TEST:** The header `<sys/types.h>` can be included more than once, in any combination  
 976 with other headers so long as the first instance of its inclusion precedes any other header that  
 977 depends upon its prior inclusion, and a symbol may be defined in more than one header with  
 978 the same value.  
 979 *Conformance for C Language Definitions: PASS*
- 980 **2.7.2 POSIX.1 Symbols**
- 981 **(IEEE Std 2003.1-1992 {4}) 05**  
 982 *UNUSED*
- 983 **5 FOR:** Headers `<aio.h>`, `<dirent.h>`, `<fcntl.h>`, `<grp.h>`, `<limits.h>`,  
 984 `<locale.h>`, `<mqueue.h>`, `<pwd.h>`, `<sched.h>`, `<semaphore.h>`,  
 985 `<signal.h>`, `<sys/mman.h>`, `<sys/stat.h>`, `<sys/times.h>`,  
 986 `<sys/wait.h>`, `<termios.h>`, `<time.h>`, `<unistd.h>` and  
 987 `<utime.h>`  
 988 **IF** the feature test macro `_POSIX_C_SOURCE` is defined to have at least the value 199309L **THEN**  
 989 **TEST:** All symbols required by IEEE Std 1003.1b-1993 to appear when a header is included  
 990 shall be made visible when the `_POSIX_C_SOURCE` feature test macro is defined.  
 991 **NOTE:** The assertion test would require an unreasonable amount of time or resources on  
 992 most implementations.  
 993 **ELSE NO\_OPTION**  
 994 *Conformance for C Language Definitions: PASS, NO\_TEST, NO\_OPTION*
- 995 **6 FOR:** `<aio.h>`, `<dirent.h>`, `<fcntl.h>`, `<grp.h>`, `<limits.h>`,  
 996 `<locale.h>`, `<mqueue.h>`, `<pwd.h>`, `<sched.h>`, `<semaphore.h>`,  
 997 `<signal.h>`, `<sys/mman.h>`, `<sys/stat.h>`, `<sys/times.h>`,  
 998 `<sys/wait.h>`, `<termios.h>`, `<time.h>`, `<unistd.h>` and `<utime.h>`  
 999 **TEST:** When a header is included, additional symbols not required or explicitly permitted by IEEE  
 1000 Std 1003.1b-1993 or the C Standard {2} to be in that header shall not be made visible,  
 1001 except when enabled by another feature test macro or by having defined  
 1002 `_POSIX_C_SOURCE` with a value larger than 199309L  
 1003 **NOTE:** The assertion test would require an unreasonable amount of time or resources on most  
 1004 implementations.  
 1005 *Conformance for C Language Definitions: PASS, NO\_TEST*
- 1006 **(IEEE Std 2003.1-1992 {4})C01**  
 1007 *UNUSED*
- 1008 **D-1 IF** the IUT supports feature test macros in addition to `_POSIX_C_SOURCE` **THEN**  
 1009 **TEST:** The PCD.1b either documents the additional feature test macros in subclause 2.7.2 or  
 1010 it does not document them at all.  
 1011 **ELSE NO\_OPTION**  
 1012 *Conformance for C Language Definitions: PASS, NO\_OPTION*
- 1013 **2.7.2.1 C Standard Language-Dependent Support**
- 1014 **7 SETUP:** A program does not use any feature test macros.  
 1015 **TEST:** The IUT makes visible only those identifiers specified as reserved identifiers in the C  
 1016 Standard {2}.  
 1017 **NOTE:** The assertion test requires setup procedures that involve an unreasonable amount of effort  
 1018 by the user of a test method.  
 1019 *Conformance for C Language Definitions: PASS, NO\_TEST*
- 1020 **8 FOR:** Each feature test macro present.  
 1021 **TEST:** The IUT makes visible only those identifiers specified by that feature test macro and those  
 1022 of the C Standard {2} when a header is included.  
 1023 **NOTE:** The assertion test requires setup procedures that involve an unreasonable amount of effort  
 1024 by the user of a test method.

1025 *Conformance for C Language Definitions: PASS, NO\_TEST*

### 1026 2.7.2.2 Common-Usage-Dependent Support

1027 **9 SETUP:** A program defines `_POSIX_C_SOURCE` before any header is included.  
 1028 **TEST:** No symbols other than those from the C Standard {2} and those made visible by feature  
 1029 test macros defined for the program (including `_POSIX_C_SOURCE`) are visible, except  
 1030 that symbols from the namespace reserved for the implementation, as defined by the C  
 1031 Standard {2}, are also permitted.  
 1032 **NOTE:** The symbols beginning with two underscores are examples of this.

1033 The assertion test requires setup procedures that involve an unreasonable amount of effort  
 1034 by the user of a test method.

1035 *Conformance for C Language Definitions: PASS, NO\_TEST*

## 1036 2.7.3 Headers and Function Prototypes

1037 (IEEE Std 2003.1-1992 {4} GA36  
 1038 *UNUSED*

1039 *M\_GA\_stdC\_proto\_decl(func\_type; function; parameters; header1; header2; header3; header4) =*  
 1040 **IF** standard **THEN**  
 1041 **SETUP:** The headers `<header1>`, `<header2>`, `<header3>`, and  
 1042 `<header 4>` are included.  
 1043 **TEST:** The function prototype *func\_type function (parameters)* is declared.  
 1044 **ELSE NO\_OPTION**

1045 **GA\_stdC\_proto\_decl**  
 1046 **FOR:** All elements except `assert ()`, `setjmp ()`, and `sigsetjmp ()`.  
 1047 *M\_GA\_stdC\_proto\_decl(func\_type; function; parameters; header1; header2; header3; header 4)*  
 1048 *Conformance for C Language Definitions: PASS, NO\_OPTION*

1049 *M\_GA\_commonC\_result\_decl(func\_type; function; header1; header2; header3; header4) =*  
 1050 **IF** the implementation does not provide C Standard {2} support **THEN**  
 1051 **SETUP:** The headers `<header1>`, `<header2>`, `<header3>`, and  
 1052 `<header 4>` are included.  
 1053 **TEST:** The function *function ()* is declared with the result type *func\_type*, or an equivalent  
 1054 type if the result type is *void*.  
 1055 **ELSE NO\_OPTION**

1056 **GA\_commonC\_result\_decl**  
 1057 **FOR:** All elements with a result type other than *int*.  
 1058 *M\_GA\_commonC\_result\_decl(func\_type; function; header1; header2; header3; header4)*  
 1059 *Conformance for C Language Definitions: PASS, NO\_OPTION*

1060 *M\_GA\_commonC\_int\_result\_decl(func-type; unction; header1; header2; header3; header4) =*  
 1061 **IF** the implementation does not provide C Standard {2} support **THEN**  
 1062 **SETUP:** The headers `<function>`, `<header1>`, `<header2>`, and  
 1063 `<header3>` are included.  
 1064 **TEST:** The interface *func\_type ()* is either declared with a result type equivalent to *int* or it  
 1065 is not declared at all.  
 1066 **ELSE NO\_OPTION**

1067 **GA\_commonC\_int\_result\_decl**  
 1068 **FOR:** All elements with a result type of *int*.  
 1069 *M\_GA\_commonC\_int\_result-decl(func\_type; function; header1; header2; header3; header4)*  
 1070 *Conformance for C Language Definitions: PASS, NO\_OPTION*

1071 **M\_GA\_setjmpDecl( )=**  
1072     **IF** the interface *setjmp()* is not defined as a macro **THEN**  
1073         **TEST:**     The function prototype *int setjmp(jmp\_buf env)* is declared with external linkage when  
1074                     the header `<setjmp.h>` is included.  
1075     **ELSE NO\_OPTION**

1076 **GA\_setjmpDecl**  
1077     **FOR:**     *setjmp()*.  
1078     **M\_GA\_setjmpDecl()**  
1079     *Conformance for C Language Definitions: PASS, NO\_OPTION*

1080 **M\_GA\_sigsetjmpDecl( )=**  
1081     **IF** the interface *sigsetjmp()* is not defined as a macro **THEN**  
1082         **TEST:**     The function prototype *int sigsetjmp( sigjmp\_buf env, int savemask)* is declared with  
1083                     external linkage when the header `<setjmp.h>` is included.  
1084     **ELSE NO\_OPTION**

1085 **GA\_sigsetjmpDecl**  
1086     **FOR:**     *sigsetjmp()*.  
1087     **M\_GA\_sigsetjmpDecl()**  
1088     *Conformance for C Language Definitions: PASS, NO\_OPTION*

1089 **M\_GA\_macro\_args(function; header1; header2; header3; header4)=**  
1090     **IF** the interface *function()* is defined as a macro **THEN**  
1091         **SETUP:**     The headers `<header1>`, `<header2>`, `<header3>`, and  
1092                     `<header4>` are included.  
1093         **TEST:**     When the macro *function()* is invoked with the correct argument types (or compatible  
1094                     argument types in the case that C Standard {2} support is provided), the macro  
1095                     evaluates its arguments only once, fully protected by parentheses when necessary,  
1096                     and protects its result value with extra parentheses when necessary.  
1097     **ELSE NO\_OPTION**

1098 **GA\_macro\_args**  
1099     **M\_GA\_macro\_args(function; header1; header2; header3; header4)**  
1100     *Conformance for C Language Definitions: PASS, NO\_OPTION*

1101 **D\_2 IF** the implementation does not provide C Standard {2} support **THEN**  
1102     **TEST:**     The PCD.1b documents in subclause 2.7.3 the equivalent constructs used when *void*  
1103                     is specified in IEEE Std 1003.1b-1993                     as a result type for a function  
1104                     prototype or it is not documented anywhere.  
1105     **ELSE NO\_OPTION**  
1106     *Conformance for C Language Definitions: PASS, NO\_OPTION*

## 1107 2.8 Numerical Limits

1108 There are no requirements for conforming implementations in this subclause.

### 1109 2.8.1 C Language Limits

1110 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.



1111 **2.8.2 Minimum Values**1112 **(IEEE Std 2003.1-1992 {4}) 02**1113 *UNUSED*

1114 **2 TEST:** The symbols in Table 2-1 shall be defined with the values shown when the header  
1115 `<limits.h>` is included.

1116 **NOTE:** This table is the same as Table 2-3 in IEEE Std 1003.1b-1993

1117 *Conformance for Numerical Limits: PASS*1118 **2.8.3 Run-Time Inceasable Values**

1119 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

1120 **2.8.4 Run-Time Invariant Values (Possibly Indeterminate)**1121 **(IEEE Std 2003.1-1992 {4}) 04**1122 *UNUSED*1123 **(IEEE Std 2003.1-1992 {4}) 05**1124 *UNUSED*1125 **(IEEE Std 2003.1-1992 {4}) 06**1126 *UNUSED*1127 **(IEEE Std 2003.1-1992 {4}) 07**1128 *UNUSED*1129 **(IEEE Std 2003.1-1992 {4}) 08**1130 *UNUSED*1131 **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 <i>exec</i> 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

	Name	Description	Value
1147	{_POSIX_NGROUPS_MAX}	The number of simultaneous supplementary group IDs per process.	0
1148	{_POSIX_OPEN_MAX}	The number of files that one process can have open at one time.	16
1149	{_POSIX_PATH_MAX}	The number of bytes in a pathname.	255
1150 1151	{_POSIX_PIPE_BUF}	The number of bytes that can be written atomically when writing to a pipe.	512
1152 1153	{_POSIX_RTSIG_MAX}	The number of realtime signal numbers reserved for application use.	8
1154	{_POSIX_SEM_NSEMS_MAX}	The number of semaphores that a process may have.	256
1155	{_POSIX_SEM_VALUE_MAX}	The maximum value a semaphore may have.	32767
1156 1157	{_POSIX_SIGQUEUE_MAX}	The number of queued signals that a process may send and have pending at the receiver(s) at any time.	32
1158	{_POSIX_SSIZE_MAX}	The value that can be stored in an object of type <i>ssize_t</i> .	32767
1159	{_POSIX_STREAM_MAX}	The number of streams that one process can have open at one time.	8
1160	{_POSIX_TIMER_MAX}	The per-process number of timers.	32
1161 1162	{_POSIX_TZNAME_MAX}	The maximum number of bytes supported for the name of a time zone (not of the TZ variable).	3

1163 (IEEE Std 2003.1-1992 {4} 09  
1164 UNUSED

1165 (IEEE Std 2003.1-1992 {4} D02  
1166 UNUSED

1167 **4 TEST:** The values defined in Table 2-2 are equal to or less than those either defined in  
1168 <limits.h> or provided by the *sysconf()* interface.  
1169 **NOTE:** This table is the same as Table 2-5 in IEEE Std 1003.1b-1993  
1170 *Conformance for Numerical Limits: PASS*

1171 **Table 2-2 – Run-Time Invariant Values (Possibly Indeterminate)**

	Name	Description	Minimum Value
1173 1174	{AIO_LISTIO_MAX}	Maximum number of I/O operations in a single list I/O call supported by the implementation.	{POSIX_AIO_LISTIO_MAX}
1175 1176	{AIO_MAX}	Maximum number of outstanding asynchronous I/O operations supported by the implementation.	{_POSIX_AIO_MAX}

	Name	Description	Minimum Value
1177 1178 1179	{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
1180 1181	{ARG_MAX}	Maximum length of arguments for the <i>exec</i> functions, in bytes, including environment data.	{_POSIX_ARG_MAX}
1182 1183	{CHILD_MAX}	Maximum number of simultaneous processes per real user ID.	{_POSIX_CHILD_MAX}
1184	{DELAYTIMER_MAX}	Maximum number of timer expiration overruns.	{_POSIX_DELAYTIMER_MAX}
1185 1186	{MQ_OPEN_MAX}	The maximum number of open message queue descriptors a process may hold.	{_POSIX_MQ_OPEN_MAX}
1187 1188	{MQ_PRIO_MAX}	The maximum number of message priorities supported by the implementation.	{POSIX_MQ_PRIO_MAX}
1189 1190	{OPEN_MAX}	Maximum number of files that one process can have open at any given time.	{_POSIX_OPEN_MAX}
1191 1192	{PAGESIZE}	Granularity in bytes of memory mapping and process memory locking.	1
1193 1194	{RTSIG_MAX}	Maximum number of realtime signals reserved for application use in this implementation.	{_POSIX_RTSIG_MAX}
1195 1196	{SEM_NSEMS_MAX}	Maximum number of semaphores that a process may have.	{_POSIX_SEM_NSEMS_MAX}
1197 1198	{SEM_VALUE_MAX}	The maximum value a semaphore may have.	{_POSIX_SEM_VALUE_MAX}
1199 1200	{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}
1201 1202	{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}
1203 1204	{TIMER_MAX}	Maximum number of timers per process supported by the implementation.	{_POSIX_TIMER_MAX}
1205 1206	{TZNAME_MAX}	The maximum number of bytes supported for the name of a time zone (not of the <i>TZ</i> variable).	{_POSIX_TZNAME_MAX}

1207 **5** **IF** a definition of one of the values in Table 2-2 is omitted from `<limits.h>`  
1208 **THEN**  
1209 **TEST:** The corresponding omitted value is equal to or greater than the stated minimum in  
1210 Table 2-2 and the actual value is provided by the *sysconf()* interface.  
1211 **ELSE NO\_OPTION**  
1212 *Conformance for Numerical Limits: PASS, NO\_OPTION*

1213 **D\_2 TEST:** The run-time invariant values for the identifiers specified in Table 2-2 are documented in  
 1214 subclause 2.8.4 of the PCD.1b.  
 1215 *Conformance for Numerical Limits: PASS*

### 1216 2.8.5 Pathname Variable Values

1217 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

### 1218 2.8.6 Invariant Values

1219 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

### 1220 2.8.7 Maximum Values

1221 **18 TEST:** The symbols in Table 2-3 shall be defined in `<limits.h>`, when it is included, and have  
 1222 the values shown.

1223 **NOTE:** This table is the same as Table 2-7a in IEEE Std 1003.1b-1993  
 1224 *Conformance for Numerical Limits: PASS*

1225 **Table 2-3 - Maximum Values**

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

## 1229 2.9 Symbolic Constants

1230 **(IEEE Std 2003.1-1992 {4} D01**  
 1231 *UNUSED*

1232 **(IEEE Std 2003.1-1992 {4} D02**  
 1233 *UNUSED*

1234 **(IEEE Std 2003.1-1992 {4} D03**  
 1235 *UNUSED*

1236 **(IEEE Std 2003.1-1992 {4} D04**  
 1237 *UNUSED*

1238 **D\_1 FOR:** Any of the symbols specified in Table 2-4 that are defined in `<unistd.h>`.

1239 **TEST:** The value associated with the symbol, the conditions under which the value may change,  
 1240 and the limits of such variations are documented in subclause 2.9 of the PCD.1b.

1241 **NOTE:** Table 2-4 is the same as Table 2-10 in IEEE Std 1003.1b-1993  
 1242 *Conformance for Symbolic Constants: PASS*

1243 **D-2 FOR:** Any of the symbols specified in Table 2-5 that are defined in `<unistd.h>`.

1244 **TEST:** The value associated with the symbol, the conditions under which the value may change,  
 1245 and the limits of such variations are documented in subclause 2.9 of the PCD.1b.

1246 **NOTE:** Table 2-5 is the same as Table 2-11 in IEEE Std 1003.1b-1993  
 1247 *Conformance for Symbolic Constants: PASS*

1248 **2.9.1 Symbolic Constants for the *access()* Function**

1249 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

1250 **2.9.2 Symbolic Constant for the *lseek()* Function**

1251 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

1252 **2.9.3 Compile-Time Symbolic Constants for Portability Specifications**1253 **(IEEE Std 2003.1-1992 {4} R02**1254 *UNUSED*1255 **(IEEE Std 2003.1-1992 {4} R03**1256 *UNUSED*1257 **1 TEST:** The values of the constants specified in Table 2-4 are not less restrictive than those  
1258 provided by the corresponding value returned by *sysconf()*.1259 *Conformance for Symbolic Constants: PASS*1260 **Table 2-4 – Compile-Time Symbolic Constants**

1261	Name	Description
1262 1263	{_POSIX_ASYNCHRONOUS_IO}	If this symbol is defined, the implementation supports the Asynchronous Input and Output option.
1264 1265	{_POSIX_FSYNC}	If this symbol is defined, the implementation supports the File Synchronization option.
1266 1267	{_POSIX_JOB_CONTROL}	If this symbol is defined, it indicates that the implementation supports the Job Control option.
1268 1269	{_POSIX_MAPPED_FILES}	If this symbol is defined, the implementation supports the Memory Mapped Files option..
1270 1271	{_POSIX_MEMLOCK}	If this symbol is defined, the implementation supports the Process Memory Locking option.
1272 1273	{_POSIX_MEMLOCK_RANGE}	If this symbol is defined, the implementation supports the Range Memory Locking option.
1274 1275	{_POSIX_MEMORY_PROTECTION}	If this symbol is defined, the implementation supports the Message Protection option.
1276 1277	{_POSIX_MESSAGE_PASSING}	If this symbol is defined, the implementation supports the Message Passing option.
1278 1279	{_POSIX_PRIORITIZED_IO}	If this symbol is defined, the implementation supports the Prioritized Input and Output option.
1280 1281	{_POSIX_PRIORITY_SCHEDULING}	If this symbol is defined, the implementation supports the Process Scheduling option.
1282 1283	{_POSIX_REALTIME_SIGNALS}	If this symbol is defined, the implementation supports the Realtime Signals Extension option.
1284 1285	{_POSIX_SAVED_IDS}	If defined, each process has a saved set-user-ID and a saved set-group-ID.

1286 1287	{_POSIX_SEMAPHORES}	If this symbol is defined, the implementation supports the Semaphores option.
1288 1289	{_POSIX_SHARED_MEMORY_OBJECTS}	If this symbol is defined, the implementation supports the Shared Memory Objects option.
1290 1291	{_POSIX_SYNCHRONIZED_IO}	If this symbol is defined, the implementation supports the Synchronized Input and Output option.
1292 1293	{_POSIX_TIMERS}	If this symbol is defined, the implementation supports the Timers option.
1294 1295	{_POSIX_VERSION}	The integer value 199309L. This value shall be used for systems that conform to this standard.

1296 **(IEEE Std 2003.1-1992 {4} 05)**  
1297 *UNUSED*

1298 **2**       **TEST:** The symbol {POSIX\_VERSION} is defined and has the value 199309L when  
1299           <unistd.h> is included.  
1300           *Conformance for Symbolic Constants: PASS*

1301 **3**       **IF** the symbol {\_POSIX\_MEMLOCK\_RANGE} is defined in <unistd.h> **THEN**  
1302           **TEST:** The symbol {\_POSIX\_MEMLOCK} shall be defined in <unistd.h>  
1303           **ELSE NO\_OPTION**  
1304           *Conformance for Symbolic Constants: PASS, NO\_OPTION*

1305 **4**       **IF** the symbol {\_POSIX\_MEMORY\_PROTECTION} is defined in <unistd.h>  
1306           **THEN**  
1307           **TEST:** At least one of the symbols {\_POSIX\_MAPPED\_FILES} or  
1308            { \_POSIX\_SHARED\_MEMORY\_OBJECTS } shall be defined in <unistd.h> .  
1309           **ELSE NO\_OPTION**  
1310           *Conformance for Symbolic Constants: PASS, NO\_OPTION*

1311 **5**       **IF** the symbol {\_POSIX\_SYNCHRONIZED\_IO} is defined in <unistd.h> **THEN**  
1312           **TEST:** The symbol {\_POSIX\_FSYNC} shall be defined in <unistd.h> .  
1313           **ELSE NO\_OPTION**  
1314           *Conformance for Symbolic Constants: PASS, NO\_OPTION*

#### 1315 **2.9.4 Execution-Time Symbolic Constants for Portability Specifications**

1316 **(IEEE Std 2003.1-1992 {4}) D05**  
1317 *UNUSED*

1318 **(IEEE Std 2003.1-1992 {4}) D06**  
1319 *UNUSED*

1320 **(IEEE Std 2003.1-1992 {4}) D07**  
1321 *UNUSED*

1322 **(IEEE Std 2003.1-1992 {4}) R04**  
1323 *UNUSED*

1324 **(IEEE Std 2003.1-1992 {4}) R05**  
1325 *UNUSED*

- 1326 (IEEE Std 2003.1-1992 {4}) R06  
1327 UNUSED
- 1328 (IEEE Std 2003.1-1992 {4}) 06  
1329 UNUSED
- 1330 (IEEE Std 2003.1-1992 {4}) 07  
1331 UNUSED
- 1332 (IEEE Std 2003.1-1992 {4}) 08  
1333 UNUSED
- 1334 **6** **TEST:** The PCD.1b documents in subclause 2.9.4 whether each of the values associated with the  
1335 symbols in Table 2-5 are defined in the header <unistd.h> and if each value defined  
1336 is -1 or other than -1.  
1337 **NOTE:** This table is the same as Table 2-11 in IEEE Std 1003.1b-1993.  
1338 *Conformance for Symbolic Constants: PASS*
- 1339 **7** **FOR:** any of the symbols in Table 2-5 that have the value -1 in the header <unistd.h>  
1340 **TEST:** The IUT shall not provide the corresponding option on any file.  
1341 **NOTE:** This table is the same as Table 2-11 in IEEE Std 1003.1b-1993.
- 1342 There is no known reliable test method for this assertion.  
1343 *Conformance for Symbolic Constants: PASS, NO\_TEST*
- 1344 **8** **FOR:** any of the symbols in Table 2-5 that have a value other than -1 in the header <unistd.h>  
1345 **TEST:** The IUT shall provide the corresponding option on all applicable files.  
1346 **NOTE:** This table is the same as Table 2-11 in IEEE Std 1003.1b-1993.
- 1347 There is no known reliable test method for this assertion.  
1348 *Conformance for Symbolic Constants: PASS, NO\_TEST*

1349 **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 <i>chown()</i> 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 POSIX.1b {3} in §7.1.1.9 can be disabled using this character value, if it is defined. See <i>tcgetattr()</i> and <i>tcsetattr()</i> .

## Section 3: Process Primitives

### 180 3.1 Process Creation and Execution

#### 181 3.1.1 Process Creation

182 Function: *fork()*

##### 183 3.1.1.1 Symbols

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

##### 185 3.1.1.2 Description

- 186 **1**       **IF** *PCTS\_sem\_init()* **THEN**  
 187           **TEST:**   Any semaphores that are open in the parent process when it makes a *fork()* call shall  
 188                           also be open in the child process.  
 189           **ELSE** *NO\_OPTION*  
 190           Conformance for fork: *PASS, NO\_OPTION*
- 191 **2**       **IF** *PCTS\_sem\_open* **THEN**  
 192           **IF** *PCTS\_GAP\_sem\_init* **THEN**  
 193               **TEST:**   Any semaphores that are open in the parent process when it makes a *fork()* call  
 194                           shall also be open in the child process.  
 195               **ELSE** *NO\_TEST\_SUPPORT*  
 196           **ELSE** *NO\_OPTION*  
 197           Conformance for fork: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 198 **3**       **FOR:**    *mlock()* and *mlockall()*  
 199       **IF** *PCTS\_function* **THEN**  
 200           **TEST:**    A child process shall not inherit any address space memory locks established by the  
 201                           parent process via calls to *function()* after a *fork()* call.  
 202           **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
 203                           assertion is to be read by substituting *function* () with the current function specified  
 204                           in the FOR clause. The name of the function also is to be substituted for each  
 205                           occurrence in the construct *PCTS\_function*  
 206           **ELSE** *NO\_OPTION*  
 207           Conformance for fork: *PASS, NO\_OPTION*
- 208 **4**       **IF** *PCTS\_mmap* **THEN**  
 209           **TEST:**    Memory mappings created in the parent are retained in the child process after a *fork()*  
 210                           call.  
 211           **ELSE** *NO\_OPTION*  
 212           Conformance for fork: *PASS, NO\_OPTION*



- 213 **5** **IF** *PCTS\_mmap* **THEN**  
 214 **TEST:** MAP\_PRIVATE mappings inherited from the parent after a *fork()* call shall also be  
 215 MAP\_PRIVATE mappings in the child, and any modifications to the data in these  
 216 mappings made by the parent prior to calling *fork()* shall be visible to the child.  
 217 **ELSE NO\_OPTION**  
 218 *Conformance for fork: PASS, NO\_OPTION*
- 219 **6** **IF** *PCTS\_mmap* **THEN**  
 220 **TEST:** Any modifications to the data in MAP\_PRIVATE mappings made by the parent after  
 221 *fork()* returns shall be visible only to the parent.  
 222 **ELSE NO\_OPTION**  
 223 *Conformance for fork: PASS, NO\_OPTION*
- 224 **7** **IF** *PCTS\_mmap* **THEN**  
 225 **TEST:** Modifications to the data in MAP\_PRIVATE mappings made by the child shall be  
 226 visible only to the child.  
 227 **ELSE NO\_OPTION**  
 228 *Conformance for fork: PASS, NO\_OPTION*
- 229 **8** **IF** *PCTS\_sched\_setscheduler1* or *PCTS\_sched\_setparam* **THEN**  
 230 **IF** *PCTS\_sched\_getscheduler* or *PCTS\_sched\_getparam* **THEN**  
 231 **TEST:** For the SCHED\_FIFO and SCHED\_RR scheduling policies, the child process shall  
 232 inherit the policy and priority settings of the parent process during a *fork()*  
 233 function.  
 234 **ELSE NO\_TEST\_SUPPORT**  
 235 **ELSE NO\_OPTION**  
 236 *Conformance for fork: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*
- 237 **9** **IF** *PCTS\_sched\_setscheduler* or *PCTS\_sched\_setparam* **THEN**  
 238 **TEST:** The PCD.1b documents the policy and priority settings on *fork()* for all scheduling  
 239 policies other than SCHED\_FIFO and SCHED\_RR in §3.1.1.2.  
 240 **ELSE NO\_OPTION**  
 241 *Conformance for fork: PASS, NO\_OPTION*
- 242 **10** **IF** *PCTS\_timer\_create* **THEN**  
 243 **IF** *PCTS\_timer\_settime* and *PCTS\_timer\_gettime* **THEN**  
 244 **TEST:** Per-process timers created by the parent are not inherited by the child process  
 245 after a *fork()* call.  
 246 **ELSE NO\_TEST\_SUPPORT**  
 247 **ELSE NO\_OPTION**  
 248 *Conformance for fork: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 249 **11** **IF** *PCTS\_mq\_open* **THEN**  
 250 **IF** *PCTS\_mq\_send* and *PCTS\_mq\_receive* **THEN**  
 251 **TEST:** A child process has its own copy of the message queue descriptors of its parent  
 252 and each of the message queue descriptors of the child refers to the same open  
 253 message queue description as the corresponding message descriptor of the  
 254 parent.  
 255 **ELSE NO\_TEST\_SUPPORT**  
 256 **ELSE NO\_OPTION**  
 257 *Conformance for fork: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 258 **12** **IF** {\_POSIX\_ASYNCHRONOUS\_IO} **THEN**  
 259 **TEST:** No asynchronous input or asynchronous output operations are inherited by the child  
 260 process after a *fork()* call.  
 261 **ELSE NO\_OPTION**  
 262 *Conformance for fork: PASS, NO\_TEST, NO\_OPTION*
- 263 **13** **FOR:** *PCTS\_aio\_read*, *PCTS\_aio\_write*, *PCTS\_lio\_listio*

264           **IF** *PCTS\_function* **THEN**  
 265               **IF** *PCTS\_aio\_cancel*, **THEN**  
 266                   **TEST:** Asynchronous input or asynchronous output operations created by calling  
 267                    *function* () are not inherited by the child process after a *fork* () call.  
 268                   **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
 269                    assertion is to be read by substituting *function*() with the current function  
 270                    specified in the FOR clause. The name of the function also is to be substituted  
 271                    for each occurrence in the construct *PCTS\_function*  
 272               **ELSE** *NO\_TEST\_SUPPORT*  
 273           **ELSE** *NO\_OPTION*  
 274            Conformance for fork: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

### 275    **3.1.1.3 Returns**

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

### 277    **3.1.1.4 Errors**

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

## 279    **3.1.2 Execute a File**

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

### 281    **3.1.2.1 Synopsis**

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

### 283    **3.1.2.1 Description**

284    **1**    **FOR:**    *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*().  
 285    **IF** *PCTS\_sem\_open* **THEN**  
 286            **SETUP:** Open a named semaphore then call *function*().  
 287            **TEST:**    Any named semaphores that are open in the calling process shall be closed as if by  
 288                    appropriate calls to *sem\_close*().  
 289            **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
 290                    assertion is to be read by substituting *function*() with the current function specified  
 291                    in the FOR clause. The name of the function also is to be substituted for each  
 292                    occurrence in the construct *PCTS\_function*.  
 293            **ELSE** *NO\_OPTION*  
 294            Conformance for *execl*, *execv*, *execle*, *execve*, *execlp*, *execvp*: *PASS, NO\_OPTION*

295    **2**    **FOR:**    *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*().  
 296    **IF** (*PCTS\_mlockall* and *PCTS\_GAP\_mlockall*) or ( *PCTS\_mlock* and *PCTS\_GAP\_mlock*) **THEN**  
 297            **SETUP:** Establish memory locks before calling *function*().  
 298            **TEST:**    Memory locks are removed after a call to *function*().  
 299            **TR:**    Establish the memory locks using as many of the interfaces *mlockall*() and *mlock*() as are  
 300                    implemented.  
 301            **NOTE:**    The interface *munlock*() can be used in the program loaded by *function* to determine  
 302                    whether or not memory locks were removed.  
 303            **ELSE** *NO\_OPTION*  
 304            Conformance for *execl*, *execv*, *execle*, *execve*, *execlp*, *execvp*: *PASS, NO\_OPTION*

305    **3**    **FOR:**    *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()  
 306    **IF** *PCTS\_mlockall* and *PCTS\_mlock* **THEN**  
 307            **IF** *PCTS\_GAP\_mlockall* and *PCTS\_GAP\_mlock* **THEN**

308                   **SETUP:** Create locked pages in the address space of the process that will call *function()*  
309                   and also map and lock the same pages into the address space of another  
310                   process.  
311                   **TEST:** The memory page locks for memory pages that are mapped into the address  
312                   space of other processes and locked by them are unaffected by a process that  
313                   has locks on those same pages and that calls *function()*.  
314                   **TR:** Establish the memory locks using as many of the interfaces *mlockall()* and *mlock()*  
315                   as are implemented.  
316                   **ELSE NO\_TEST\_SUPPORT**  
317                   **ELSE NO\_OPTION**  
318                   Conformance for *execl*, *execv*, *execle*, *execve*, *execlp*, *execvp*: *PASS*, *NO\_TEST\_SUPPORT*,  
319                   *NO\_OPTION*

320   **D\_1 FOR:** *execl()*, *execv()*, *execle()*, *execve()*, *execlp()*, *execvp()*  
321                   **IF** *PCTS\_mmap* or *PCTS\_shm\_open* and a PCD.1b documents the following **THEN**  
322                   **TEST:** A PCD.1b that documents the effect on memory locks in the *function()* fails does so  
323                   in §3.1.2.2.  
324                   **ELSE NO\_OPTION**  
325                   Conformance for *execl*, *execv*, *execle*, *execve*, *execlp*, *execvp*: *PASS*, *NO\_OPTION*

326   **4 FOR:** *execl()*, *execv()*, *execle()*, *execve()*, *execlp()*, *execvp()*  
327                   **IF** *PCTS\_mmap* or *PCTS\_shm\_open* **THEN**  
328                   **TEST:** Memory mappings created in a process are unmapped before the address space is  
329                   rebuilt for the new process image after a call to *function()*.  
330                   **NOTE:** There is no known portable test method for this assertion.  
331                   **ELSE NO\_OPTION**  
332                   Conformance for *execl*, *execv*, *execle*, *execve*, *execlp*, *execvp*: *PASS*, *NO\_TEST*, *NO\_OPTION*

333   **5 FOR:** *execl()*, *execv()*, *execle()*, *execve()*, *execlp()*, *execvp()*  
334                   **IF** *PCTS\_sched\_setscheduler* and *PCTS\_getscheduler* **THEN**  
335                   **TEST:** The policy and priority settings for the *SCHED\_FIFO* and *SCHED\_RR* scheduling  
336                   policies are not changed for a process that calls *function()*.  
337                   **ELSE NO\_OPTION**  
338                   Conformance for *execl*, *execv*, *execle*, *execve*, *execlp*, *execvp*: *PASS*, *NO\_OPTION*

339   **D\_2 FOR:** *execl()*, *execv()*, *execle()*, *execve()*, *execlp()*, *execvp()*  
340                   **IF** *PCTS\_sched\_setscheduler* and *PCTS\_getscheduler* **THEN**  
341                   **TEST:** The PCD.1b documents for scheduling policies other than *SCHED\_FIFO* and *SCHED\_RR*,  
342                   the policy and priority settings after a call to *function()* in §3.1.2.2.  
343                   **ELSE NO\_OPTION**  
344                   Conformance for *execl*, *execv*, *execle*, *execve*, *execlp*, *execvp*: *PASS*, *NO\_OPTION*

345   **6 FOR:** *execl()*, *execv()*, *execle()*, *execve()*, *execlp()*, *execvp()*  
346                   **IF** *PCTS\_timer\_create* and *PCTS\_timer\_settime* **THEN**  
347                   **TEST:** Per-process timers created by the calling process are deleted before replacing the  
348                   current process image with the new process image after a call to *function()*.  
349                   **NOTE:** There is no known portable test method for this assertion.  
350                   **ELSE NO\_OPTION**  
351                   Conformance for *execl*, *execv*, *execle*, *execve*, *execlp*, *execvp*: *PASS*, *NO\_TEST*, *NO\_OPTION*

352   **7 FOR:** *execl()*, *execv()*, *execle()*, *execve()*, *execlp()*, *execvp()*  
353                   **IF** *PCTS\_timer\_create* and *PCTS\_timer\_settime* **THEN**  
354                   **TEST:** Per-process timers created by the calling process are deleted after a call to *function()*.  
355                   **NOTE:** This can be tested by setting a timer before the *function()* call and waiting in the new  
356                   process image for the timer's signal. If no signal arrives after a sufficiently long  
357                   time, the timer was destroyed or the equivalent. To determine of the timer was  
358                   actually destroyed, try to create {*TIMER\_MAX*} timers. If this is possible then the  
359                   timer was destroyed.  
360                   **ELSE NO\_OPTION**

361 *Conformance for execl, execv, execl, execve, execlp, execvp: PASS, NO\_OPTION*

362 **8** **FOR:** *execl(), execv(), execl(), execve(), execlp(), execvp()*

363 **IF** *PCTS\_mq\_open* **THEN**

364 **IF** *PCTS\_mq\_close* and *PCTS\_mq\_send* and *PCTS\_mq\_receive* **THEN**

365 **SETUP:** Create two processes, one will call *function()* and the other will test the

366 message queue with the new process image created by *function()*.

367 **TEST:** After a call to *function()*, all open message queue descriptors in the calling

368 process shall be closed, the association between the message queue descriptor

369 and the message queue is removed.

370 **TR:** Test for at least two message queues.

371 **NOTE:** This may not be testable if the use of message queue descriptors is really

372 undefined after a close.

373 **ELSE** *NO\_TEST\_SUPPORT*

374 **ELSE** *NO\_OPTION*

375 *Conformance for execl, execv, execl, execve, execlp, execvp: PASS, NO\_TEST\_SUPPORT,*

376 *NO\_OPTION*

377 **9** **FOR:** *execl(), execv(), execl(), execve(), execlp(), execvp()*

378 **IF** *PCTS\_mq\_open* **THEN**

379 **IF** (*{\_POSIX\_MESSAGE\_PASSING}* and *{\_POSIX\_REALTIME\_SIGNALS}*) or (*PCTS\_mq\_close* and

380 *PCTS\_mq\_send* and *PCTS\_mq\_notify*) **THEN**

381 **SETUP:** Create two processes, one will call *function()* and the other will test the

382 message queue with the new process image created by *function()*.

383 **TEST:** After a call to *function()*, all open message queue descriptors in the calling

384 process shall be closed, an attached message queue notification request is

385 removed and the message queue is available for another process to attach a

386 notification.

387 **NOTE:** This may not be testable if the use of message queue descriptors is really

388 undefined after a close.

389 **ELSE** *NO\_TEST\_SUPPORT*

390 **ELSE** *NO\_OPTION*

391 *Conformance for execl, execv, execl, execve, execlp, execvp: PASS, NO\_TEST\_SUPPORT,*

392 *NO\_OPTION*

393 **10** **FOR:** *execl(), execv(), execl(), execve(), execlp(), execvp()*

394 **IF** *{\_POSIX\_ASYNCHRONOUS\_IO}* **THEN**

395 **TEST:** Any asynchronous I/O operations that are not canceled after calling *function()*

396 complete as if the *function()* call had not yet occurred, but any associated signal

397 notifications are suppressed.

398 **ELSE** *NO\_OPTION*

399 *Conformance for execl, execv, execl, execve, execlp, execvp: PASS, NO\_TEST, NO\_OPTION*

400 **D\_3** **FOR:** *execl(), execv(), execl(), execve(), execlp(), execvp()*

401 **IF** *{\_POSIX\_ASYNCHRONOUS\_IO}* and a PCD.1b documents the following **THEN**

402 **TEST:** A PCD.1b that documents whether the *function()* itself blocks awaiting asynchronous

403 I/O completion does so in §3.1.2.2.

404 **ELSE** *NO\_OPTION*

405 *Conformance for execl, execv, execl, execve, execlp, execvp: PASS, NO\_OPTION*

406 **11** **FOR:** *execl(), execv(), execl(), execve(), execlp(), execvp()*

407 **IF** *{\_POSIX\_ASYNCHRONOUS\_IO}* **THEN**

408 **TEST:** The new process image created after an *function()* call is not affected by the presence

409 of outstanding asynchronous I/O operations at the time the *function()* function is

410 called.

411 **ELSE** *NO\_OPTION*

412 *Conformance for execl, execv, execl, execve, execlp, execvp: PASS, NO\_TEST, NO\_OPTION*

413 **D\_4** **FOR:** *execl(), execv(), execl(), execve(), execlp(), execvp()*

414           **IF** {\_POSIX\_ASYNCHRONOUS\_IO} **THEN**  
 415                 **TEST:**     The PCD.1b documents whether any asynchronous I/O operation is canceled, and  
 416                                 which I/O may be canceled upon a call to *function()*, in §3.1.2.2.  
 417           **ELSE NO\_OPTION**  
 418                 *Conformance for execl, execv, execl, execve, execlp, execvp: PASS, NO\_OPTION*

### 419   **3.1.2.3 Returns**

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

### 421   **3.1.2.4 Errors**

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

## 423   **3.2 Process Termination**

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

### 425   **3.2.1 Wait for Process Termination**

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

#### 427   **3.2.1.1 Synopsis**

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

#### 429   **3.2.1.2 Description**

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

#### 431   **3.2.1.3 Returns**

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

#### 433   **3.2.1.4 Errors**

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

### 435   **3.2.2 Terminate a Process**

436   Function: *\_exit()*

#### 437   **3.2.2.1 Synopsis**

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

#### 439   **3.2.2.2 Description**

440   **1**           **IF** *PCTS\_sem\_open* **THEN**  
 441                 **IF** *PCTS\_sem\_trywait* and *PCTS\_sem\_getvalue* **THEN**  
 442                         **SETUP:**   Create {PCTS\_SEM\_NSEMS\_MAX} named semaphores with large initial values  
 443                                 and lock each one at least once.  
 444                         **TEST:**     After a call to *\_exit()*, all open named semaphores in the calling process has no  
 445                                 effect on the state of such semaphores.  
 446                 **ELSE NO\_TEST\_SUPPORT**  
 447                 **ELSE NO\_OPTION**

448 *Conformance for \_exit: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

449 **2** **FOR:** *mlock()* and *mlockall()*

450 **IF** *PCTS\_function()* **THEN**

451 **IF** *PCTS\_GAP\_function()* **THEN**

452 **SETUP:** Lock all pages of the process in memory.

453 **TEST:** Any memory locks established by the process via calls to *function()* are

454 removed after a call to *\_exec()*.

455 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The

456 assertion is to be read by substituting *function()* with the current function

457 specified in the FOR clause. The name of the function also is to be substituted

458 for each occurrence in the construct *PCTS\_function*.

459 **ELSE** *NO\_TEST\_SUPPORT*

460 **ELSE** *NO\_OPTION*

461 *Conformance for \_exit: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

462 **3** **FOR:** *mlock()* and *mlockall()*

463 **IF** *PCTS\_function()* **THEN**

464 **IF** *PCTS\_GAP\_function()* **THEN**

465 **SETUP:** Use *function()* to create locked pages in the address space of the process calling

466 *\_exit()* that are also mapped into the address spaces of other processes and are

467 locked by them.

468 **TEST:** The memory locks established by the other processes are unaffected by the call

469 by this process to *-exit()*.

470 **TR:** Test for at least two other processes.

471 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The

472 assertion is to be read by substituting *function()* with the current function

473 specified in the FOR clause. The name of the function also is to be substituted

474 for each occurrence in the construct *PCTS\_function*

475 **ELSE** *NO\_TEST\_SUPPORT*

476 **ELSE** *NO\_OPTION*

477 *Conformance for \_exit: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

478 **4** **IF** *PCTS\_mmap* **THEN**

479 **TEST:** Memory mappings created in the process are unmapped before the process is

480 destroyed after a call to *\_exec()*.

481 **ELSE** *NO\_OPTION*

482 *Conformance for \_exit: PASS, NO\_TEST, NO\_OPTION*

483 **5** **IF** *PCTS\_mq\_open* **THEN**

484 **IF** *PLCTS\_mq\_notify* **THEN**

485 **SETUP:** Create a message queue, a process to call *exit()*, and another process to check

486 on the state of the message queue. Call *mq\_notify()* from the process that will

487 call *\_exit()*.

488 **TEST:** All open message queue descriptors in the process calling *\_exit()* are closed

489 which allows other processes to issue successful *mq\_notify()* calls.

490 **ELSE** *NO\_TEST\_SUPPORT*

491 **ELSE** *NO\_OPTION*

492 *Conformance for \_exit: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

493 **6** **IF** *{\_POSIX\_ASYNCHRONOUS\_IO}* **THEN**

494 **TEST:** Those asynchronous I/O operations that are not canceled after a call to *\_exit()*

495 completes as if the *\_exit()* operation had not yet occurred, but any associated signal

496 notifications are suppressed.

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

498 **ELSE** *NO\_OPTION*

499 *Conformance for \_exit: PASS, NO\_TEST, NO\_OPTION*

500 **D\_1** **IF** a PCD.1b documents the following **THEN**

501                   **TEST:**    A PCD.1b that documents whether or not the `_exit()` operation itself blocks  
 502   awaiting the completion of asynchronous I/O does so in §3.2.2.2.  
 503                   **ELSE NO\_OPTION**  
 504                   *Conformance for `_exit`: PASS, NO\_OPTION*

505 **D\_2 TEST:**    The PCD.1b documents whether any asynchronous I/O is canceled, and which asynchronous I/O  
 506   may be canceled upon a call to `_exit()` in §3.2.2.  
 507                   *Conformance for `_exit`: PASS*

### 508 3.2.2.3 Returns

509 There are no requirements for conforming implementations in this subclause.

## 510 3.3 Signals

### 512 3.3.1 Signal Concepts

#### 513 3.3.1.1 Signal Names

514 NOTE: The tables 3-1, 3-2, and 3-3 are kept in the same order in this subclause as in POSIX.1b {3} so as to facilitate  
 515 the reader's following the correspondence between assertions in this standard and the requirements specified in  
 516 POSIX.1b {3}.

517 **Table 3-1 – Required Signals**

518	Symbolic Constant	Default Action	Description
520	SIGABRT	1	Abnormal termination signal, such as is initiated by the <code>abort()</code> function.
521	SIGALRM	1	Timeout signal, such as initiated by the <code>alarm()</code> function.
522	SIGFPE	1	Erroneous arithmetic operations, such as division by zero or an operation resulting in overflow.
523			
524	SIGHUP	1	Hangup detected on controlling terminal or death of controlling process.
525			
526	SIGILL	1	Detection of an invalid hardware instruction.
527			
528	SIGINT	1	Interactive attention signal.
529	SIGKILL	1	Termination signal (cannot be caught or ignored).
530	SIGPIPE	1	Write on a pipe with no readers.
531	SIGQUIT	1	Interactive termination signal.
532	SIGSEGV	1	Detection of an invalid memory reference.
533	SIGTERM	1	Termination signal.
534	SIGUSR1	1	Reserved as application-defined signal 1.
535	SIGUSR2	1	Reserved as application-defined signal 2.

536 NOTE:    The default actions are

537           1    Abnormal termination of the process.

538 **1        SETUP:**    Include the header `<signal.h>`.

539 **TEST:**    The constants shown in Table 3-3 are defined.  
 540 *Conformance for `signal.h`: PASS*

541 **2** **IF** {\_POSIX\_MEMORY\_PROTECTION} **THEN**  
 542 **SETUP:** Include the header <signal.h>.  
 543 **TEST:** The signals shown in Table 3-3 behave with the specified default action.

544 **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.

553 **NOTE:** The default actions are  
 554 2 Ignore the signal.  
 555 3 Stop the process.  
 556 4 Continue the process if it is currently stopped; otherwise, ignore the signal.

557 **Table 3-3 - Memory Protection Signals**

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

561 **TR:** Test for child processes and processes that are not children of the calling process.  
 562 **ELSE NO\_OPTION**  
 563 *Conformance for signal.h: PASS, NO\_OPTION*

564 **3** **SETUP:** Include the header <signal.h> .  
 565 **TEST:** The macros SIGRTMIN and SIGRTMAX are defined and evaluate to integral expressions.  
 566 *Conformance for signal.h: PASS*

567 **4** **SETUP:** Include the header <signal.h> .  
 568 **TEST:** The signal numbers in this range SIGRTMIN to SIGRTMAX do not overlap with any of the  
 569 signals specified in Tables 3-1, 3-2, or 3-3.  
 570 *Conformance for signal.h: PASS*

571 **5** **SETUP:** Include the header <signal.h> .  
 572 **TEST:** The range SIGRTMIN through SIGRTMAX inclusive includes at least {RTSIG\_MAX} signal  
 573 numbers.  
 574 *Conformance for signal.h: PASS*

575 **D\_1 TEST:** The PCD.1b documents whether the realtime signal behavior for the queuing of signals and the  
 576 passing of application defined values is supported for each of the signals defined in Tables 3-1,  
 577 3-2, and 3-3 in §3.3.1.1.  
 578 *Conformance for signal.h: PASS*



579 **3.3.1.2 Signal Generation and Delivery**

580 **6**       **SETUP:** Include the header `<signal.h>`.  
 581       **TEST:** The *sigevent* structure is defined, *sigev\_value* is equal to or greater than SIGRTMIN or less  
 582       than SIGRTMAX, and contains at least the following members:

Member Type	Member Name	Description
<i>int</i>	<i>sigev_notify</i>	Notification type
<i>int</i>	<i>sigev_signo</i>	Signal number
<i>union sigval</i>	<i>sigev_value</i>	Signal value

588       *Conformance for signal.h: PASS*

589 **D\_2** **IF** a PCD.1b documents the following **THEN**  
 590       **TEST:** A PCD.1b that documents any extensions that are added to the *sigevent* structure and  
 591       how they are enabled does so in §3.3.1.2.  
 592       **TR:** Only extensions as permitted in 1.3.1.1 item (2) in IEEE Std 1003.1b-1993       may  
 593       be added.  
 594       **ELSE NO\_OPTION**  
 595       *Conformance for signal.h: PASS, NO\_OPTION*

596 **7**       **SETUP:** Include the header `<signal.h>`.  
 597       **TEST:** The symbols SIGEV\_NONE and SIGEV\_SIGNAL are defined.  
 598       *Conformance for signal.h: PASS*

599 **D\_3** **IF** a PCD.1b documents the following **THEN**  
 600       **TEST:** A PCD.1b that documents additional notification mechanisms to use when  
 601       asynchronous events occur does so in §3.3.1.2.  
 602       **ELSE NO\_OPTION**  
 603       *Conformance for signal.h: PASS, NO\_OPTION*

604 **M\_GA\_sigev\_value() =**  
 605       **IF** {\_POSIX\_REALTIME\_SIGNALS} **THEN**  
 606       **TEST:** The *sigev\_value* member of the *sigevent* structure is passed to a signal-catching  
 607       function at the time of the signal delivery as the *si\_value*- member of the *siginfo\_t*  
 608       structure.  
 609       **TR:** Test for all catchable signals.  
 610       **ELSE NO\_OPTION**

611 **GA\_sigev\_value**  
 612       **FOR:** *lio\_listio()*, *timer\_create()*, *mq\_notify()*, *aio\_read()*, *aio\_write()*, and *aio\_fsync()*  
 613       **M\_GA\_sigev\_value ()**  
 614       *Conformance for signal.h: PASS, NO\_OPTION*

615 **M\_GA\_sigqueueValue() =**  
 616       **IF** {\_POSIX\_REALTIME\_SIGNALS} **THEN**  
 617       **TEST:** The *value* parameter to *sigqueue()* is passed to a signal-catching function at the time  
 618       of the signal delivery as the *si\_value* member of the *siginfo\_t* structure.  
 619       **TR:** Test for all catchable signals.  
 620       **ELSE NO\_OPTION**

621 **GA\_sigqueueValue**  
 622       **FOR:** *sigqueue()*  
 623       **M\_GA\_sigqueueValue()**  
 624       *Conformance for signal.h: PASS, NO\_OPTION*

625 **8**       **SETUP:** Include the header `<signal.h>`.  
 626           **TEST:**    The *signal* is defined and contains at least the following members:

	<b>Member Type</b>	<b>Member Name</b>	<b>Description</b>
627			
628			
629	<i>int</i>	<i>sival_int</i>	Integer signal value
630	<i>void*</i>	<i>sival_ptr</i>	Pointer signal value

631           *Conformance for signal.h: PASS*

632 ***M\_GA\_sigPending(function)=***

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

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

638       **TR:** Test for all catchable signals both with SA\_SIGINFO set and not set.

639 **GA\_sigPending**

640       **FOR:**    *sigqueue()*, *lio\_listio()*, *timer\_create()*, *mq\_notify()*, *aio\_read()*, *aio\_write()*, and  
 641 *aio\_fsync()*

642       ***M\_GA\_sigPending(function)***

643       *Conformance for signal.h: PASS, NO\_OPTION*

644 ***M\_GA\_sigPendingQueued(function)=***

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

647       **TEST:**    Multiple occurrences of signals is generated by the *function()* interface with the SA\_SIGINFO  
 648 flag set for that signal will be queued in FIFO order.

649       **TR:** Test for all catchable signals.

650 **GA\_sigPendingQueued**

651       **FOR:**    *sigqueue()*, *lio\_listio()*, *timer\_create()*, *mq\_notify()*, *aio\_read()*, *aio\_write()*, and  
 652 *aio\_fsync()*

653 **D\_4 IF** a PCD.1b documents the following **THEN**

654       **TEST:**    A PCD.1b that documents whether signals generated by the *sigqueue()* function or any  
 655 signal-generating function that supports the specification of an application-defined  
 656 value are queued when the SA\_SIGINFO flag is not set for that signal does so in  
 657 §3.3.1.2.

658       **ELSE NO\_OPTION**

659       *Conformance for signal.h: PASS, NO\_OPTION*

660 ***M\_GA\_queuedAndRegularSignals(function)=***

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

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

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

667 **GA\_queuedAndRegularSignals**

668       **FOR:**    *sigqueue()*, *lio\_listio()*, *timer\_create()*, *mq\_notify()*, *aio\_read()*, *aio\_write()*, and  
 669 *aio\_fsync()*

670       ***M\_GA\_queuedAndRegularSignals(function)***

671       *Conformance for signal.h: PASS, NO\_OPTION*

672 **9**       **IF** `_POSIX_REALTIME_SIGNALS` **THEN**

673                   **IF** *PCTS\_sigqueue* **THEN**  
 674                    **SETUP:** Create multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, that  
 675                    are pending.  
 676                    **TEST:** The implementation delivers the pending unblocked signal with the lowest  
 677                    signal number within that range.  
 678                    **TR:** Test with each signal in the range being the lowest unblocked signal.  
 679                    **ELSE NO\_TEST\_SUPPORT**  
 680                    **ELSE NO\_OPTION**  
 681                    *Conformance for signal.h: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

682   **10**           **IF** *\_POSIX\_REALTIME\_SIGNALS* **THEN**  
 683                    **IF** *PCTS\_sigqueue* **THEN**  
 684                    **SETUP:** Create a pending signal and additional signals queued to the same signal  
 685                    number.  
 686                    **TEST:** After a pending signal is delivered, the signal shall remain pending until all  
 687                    queued signals have been delivered at which time the pending indication is  
 688                    reset.  
 689                    **TR:** Test for all catchable signals.  
 690                    **ELSE NO\_TEST\_SUPPORT**  
 691                    **ELSE NO\_OPTION**  
 692                    *Conformance for signal.h: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

693   **11**           **IF** *\_POSIX\_REALTIME\_SIGNALS* **THEN**  
 694                    **IF** *PCTS\_sigqueue* **THEN**  
 695                    **SETUP:** Create a pending signal.  
 696                    **TEST:** After a pending signal is delivered and there are no queued signals to the same  
 697                    signal number, the pending indication is reset.  
 698                    **TR:** Test for all catchable signals.  
 699                    **ELSE NO\_TEST\_SUPPORT**  
 700                    **ELSE NO\_OPTION**  
 701                    *Conformance for signal.h: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

### 702   **3.3.1.3 Signal Actions**

703   **12**           **IF** *{\_POSIX\_REALTIME\_SIGNALS}* **THEN**  
 704                    **IF** *PCTS\_sigqueue* **THEN**  
 705                    **TEST:** The default action for the realtime signals in the range of SIGRTMIN through  
 706                    SIGRTMAX is to terminate the process abnormally.  
 707                    **TR:** Test for all signals in the range.  
 708                    **ELSE NO\_TEST\_SUPPORT**  
 709                    **ELSE NO\_OPTION**  
 710                    *Conformance for signal.h: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

711   **13**           **IF** *{\_POSIX\_JOB\_CONTROL}* and *{\_POSIX\_REALTIME\_SIGNALS}* **THEN**  
 712                    **TEST:** Setting a signal action to SIG\_DEL for a SIGCHLD signal that is pending causes the  
 713                    pending signal to be discarded, whether or not it is blocked; in addition, any queued  
 714                    values pending are discarded.  
 715                    **ELSE NO\_OPTION**  
 716                    *Conformance for signal.h: PASS, NO\_OPTION*

717   **14**           **IF** *{\_POSIX\_JOB\_CONTROL}* and *{\_POSIX\_REALTIME\_SIGNALS}* **THEN**  
 718                    **TEST:** Setting a signal action to SIG\_DFL for a SIGCHLD signal that is pending causes the  
 719                    resources used to queue any pending signals and values to be released and made  
 720                    available to queue other signals.  
 721                    **NOTE:** There is no known portable test method for this assertion.  
 722                    **ELSE NO\_OPTION**  
 723                    *Conformance for signal.h: PASS, NO\_TEST, NO\_OPTION*

724 **D\_5 IF** a PCD.1b documents the following **THEN**  
725       **TEST:** A PCD.1b that documents the behavior of a process after it ignores a SIGFPE, SIGILL,  
726               SIGSEGV, or SIGBUS signal that was not generated by the *kill()* function, the *sigqueue()*  
727               function, or the *raise()* function as defined by the C Standard {2} does so in §3.3.1.3.  
728       **ELSE NO\_OPTION**  
729       *Conformance for signal.h: PASS, NO\_OPTION*

730 **15 IF** {\_POSIX\_REALTIME\_SIGNALS} **THEN**  
731       **IF** PCTS\_ sigqueue **THEN**  
732           **TEST:** Setting a signal action to SIG\_IGN for a signal that is pending causes the  
733           pending signal to be discarded, whether or not it is blocked; in addition, any  
734           queued values pending are discarded.  
735           **TR:** Test for all catchable signals consistent with the implemented options.  
736       **ELSE NO\_TEST\_SUPPORT**  
737       **ELSE NO\_OPTION**  
738       *Conformance for signal.h: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

739 **16 IF** {\_POSIX\_REALTIME\_SIGNALS} **THEN**  
740       **IF** PCTS\_ sigqueue **THEN**  
741           **TEST:** If the implementation limits the number of outstanding queued signals to  
742           {SIGQUEUE\_MAX}, this test can verify that signal can be sent after setting the  
743           action to SIG\_IGN in the target process. Setting a signal action to SIG\_IGN for a  
744           signal that is pending causes the resources used to queue the pending signal and  
745           its associated value to be released and made available to queue other signals.  
746           **NOTE:** There is no known portable test method for this assertion.  
747       **ELSE NO\_TEST\_SUPPORT**  
748       **ELSE NO\_OPTION**  
749       *Conformance for signal.h: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

750 **17 IF** {\_POSIX\_MEMORY\_PROTECTION} **THEN**  
751       **IF** PCTS\_ sigqueue **THEN**  
752           **TEST:** On delivery of a signal specified in Table 3-3, the receiving process executes  
753           the signal-catching function at the specified address.  
754           **TR:** Test for all signals in Table 3-3.  
755       **ELSE NO\_TEST\_SUPPORT**  
756       **ELSE NO\_OPTION**  
757       *Conformance for signal.h: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

758 **18 IF** PCTS\_ sigqueue **THEN**  
759       **SETUP:** Clear the SA\_SIGINFO flag and establish a signal-catching function based on the  
760       following prototype:  
761                               void func(int signo);

762       **TEST:** The signal-catching function is passed *signo* as a parameter.  
763       **TR:** Test for all catchable signals consistent with the implemented options.  
764       **ELSE NO\_OPTION**  
765       *Conformance for signal.h: PASS, NO\_OPTION*

766 **19 IF PCTS\_sigqueue THEN**  
 767 **SETUP:** Set the SA\_SIGINFO flag and establish a signal-catching function based on the  
 768 following prototype:  
 769 

```
void func(int signo, siginfo_t *info,  

  770 void *context);
```

  
 771 **TEST:** The signal-catching function is passed *signo* (the signal number of the signal  
 772 being delivered) and *info* (a pointer to a *siginfo\_t* structure).  
 773 **TR:** Test for all catchable signals consistent with the implemented options.  
 774 **ELSE NO\_OPTION**  
 775 *Conformance for signal.h: PASS, NO\_OPTION*

776 **20 SETUP:** Include the header `<signal.h>`.  
 777 **TEST:** A structure type *siginfo\_t* is defined and contains at least the following  
 778 members:

	Member Type	Member Name	Description
779	<i>int</i>	<i>si_signo</i>	Signal number
780	<i>int</i>	<i>si_code</i>	Cause of the signal
781	<i>union sigval</i>	<i>si_value</i>	Signal value

784 *Conformance for signal.h: PASS*

785 **21 IF PCTS\_sigqueue THEN**  
 786 **SETUP:** With the SA\_SIGINFO flag set, create a signal-catching function. Then generate a  
 787 signal for that signal-catching function to handle.  
 788 **TEST:** The *si\_signo* member of the *siginfo\_t* structure contains the signal number and it is  
 789 the same as the *signo* parameter passed to the signal-catching function.  
 790 **TR:** Test for all catchable signals consistent with implemented options.  
 791 **ELSE NO\_OPTION**  
 792 *Conformance for signal.h: PASS, NO\_OPTION*

793 **22 SETUP:** With the SA\_SIGINFO flag set, create a signal-catching function and set an application-  
 794 specific value to be passed to it. Then generate a signal for that signal-catching function  
 795 to handle by calling the *kill()* function.  
 796 **TEST:** The *si\_code* member of the *siginfo\_t* structure contains SI\_USER and the *si\_signo* member  
 797 of the *siginfo\_t* structure contains the signal number and it is the same as the *signo*  
 798 parameter passed to the signal-catching function.  
 799 *Conformance for signal.h: PASS*

800 **D\_6 IF** a PCD.1b documents the following **THEN**  
 801 **TEST:** A PCD.1b that documents whether the *si\_code* number of the *siginfo\_t* structure is set  
 802 to SI\_USER if the signal was sent by the *raise()* or *abort()* functions as defined in the  
 803 C Standard {2} or any similar functions provided as implementation extensions does  
 804 so in §3.3.1.3.  
 805 **ELSE NO\_OPTION**  
 806 *Conformance for signal.h: PASS, NO\_OPTION*

807 **23 IF PCTS\_SIGQUEUE THEN**  
 808 **SETUP:** With the SA\_SIGINFO flag set, declare a signal-catching function and choose an  
 809 application-specific value to be passed to it. Then generate a signal for that signal-  
 810 catching function to handle by calling the SI\_QUEUE function.  
 811 **TEST:** The *si\_code* member of the *siginfo\_t* structure contains SI\_QUEUE, a code identifying  
 812 the cause of the signal as being generated by the *sigqueue()* function, and the *si\_signo*

813 member of the *siginfo\_t* structure contains the signal number and it is the same as the  
814 *signo* parameter passed to the signal-catching function.  
815 **ELSE NO\_OPTION**  
816 *Conformance for signal.h: PASS, NO\_OPTION*

817 **24 IF PCTS\_timer\_settime THEN**  
818 **IF PCTS\_timer\_create THEN**  
819 **SETUP:** With the SA\_SIGINFO flag set, declare a signal-catching function and choose  
820 an application-specific value to be passed to it. Then generate a signal for that  
821 signal-catching function to handle by setting up a timer expiration for a timer  
822 set by *timer\_settime()*.  
823 **TEST:** The *se\_code* member of the *siginfo\_t* structure contains SI\_TIMER, a code  
824 identifying the cause of the signal generation as the expiration of a timer set by  
825 *timer\_settime()*, and the *si\_signo* member of the *siginfo\_t* structure contains the  
826 signal number and it is the same as the *signo* parameter passed to the signal-  
827 catching function.  
828 **ELSE NO\_TEST\_SUPPORT**  
829 **ELSE NO\_OPTION**  
830 *Conformance for signal.h: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

831 **25 IF {\_POSIX\_ASYNCHRONOUS\_IO} and {POSIX\_REALTIME\_SIGNALS} THEN**  
832 **SETUP:** With the SA\_SIGINFO flag set, declare a signal-catching function and choose an  
833 application-specific value to be passed to it. Then generate a signal for that signal-  
834 catching function to handle by completing an asynchronous I/O request.  
835 **TEST:** The *si\_code* member of the *siginfo\_t* structure contains SI\_ASYNCIO, a code  
836 identifying the cause of the signal generation as the completion of an asynchronous  
837 I/O request, and the *si\_signo* member of the *siginfo\_t* structure contains the signal  
838 number and it is the same as the *signo* parameter passed to the signal-catching  
839 function.  
840 **ELSE NO\_OPTION**  
841 *Conformance for signal.h: PASS, NO\_OPTION*

842 **26 IF {\_POSIX\_MESSAGE\_PASSING} and {\_POSIX\_REALTIME\_SIGNALS} THEN**  
843 **SETUP:** With the SA\_SIGINFO flag set, declare a signal-catching function and choose an  
844 application-specific value to be passed to it. Then generate a signal for that signal-  
845 catching function to handle by making a message arrive on an empty message queue.  
846 **TEST:** The *si\_code* member of the *siginfo\_t* structure contains the code SI\_MESGQ, which  
847 means the signal was generated by the arrival of a message on an empty message  
848 queue, and the *si\_signo* member of the *siginfo\_t* structure contains the signal number  
849 and it is the same as the *signo* parameter passed to the signal-catching function.  
850 **ELSE NO\_OPTION**  
851 *Conformance for signal.h: PASS, NO\_OPTION*

852 **D\_7 TEST:** The PCD.1b documents the implementation-defined value, which is not equal to any of the values  
853 SI\_USER, SI\_QUEUE, SI\_TIMER, SI\_ASYNCIO, and SI\_MESGQ, and that is set in the *si\_code* if a signal  
854 was not generated by one of the following functions or events:

- 855 1. The *kill()* function.
- 856 2. The *raise()* or *abort()* functions as defined in the C Standard {2}, if they set  
857 *si\_code* to SI\_USER.
- 858 3. The *sigqueue()* function.
- 859 4. The *timer\_settime()* function.
- 860 5. The completion of an asynchronous I/O request.
- 861 6. The arrival of a message on an empty message queue in §3.3.1.3.

862 *Conformance for signal.h: PASS*

863 **27** **IF** *PCTS\_MORE\_SA\_SIGINFO\_SIGNALS* and *{\_POSIX\_REALTIME\_SIGNALS}* **THEN**  
864 **SETUP:** Establish a signal-catching function with the *SA\_SIGINFO* flag set. Then generate that  
865 signal by a means other than calling *kill()*, *raise()* and *abort()* (if they set to  
866 *sigqueue()*, or *timer\_settime()* or by completion of an asynchronous I/O request or by  
867 the arrival of a message on an empty message queue.  
868 **TEST:** The *si\_code* is set to an implementation-defined value that is not equal to any of the  
869 values defined in IEEE Std 1003.1b-1993 for *si\_code*.  
870 **TR:** Test for each such implementation-defined value.  
871 **NOTE:** It is possible to perform this test by calling a target system-specific function that  
872 contains the tests.  
873 **ELSE NO\_OPTION**  
874 *Conformance for signal.h: PASS, NO\_OPTION*

875 **28** **IF** *{\_POSIX\_REALTIME\_SIGNALS}* **THEN**  
876 **IF** *PCTS\_sigqueue* **THEN**  
877 **SETUP:** Create a signal so that *si\_code* contains the value *SI\_QUEUE*.  
878 **TEST:** The *si\_value* contains the application-specified signal value.  
879 **ELSE NO\_TEST\_SUPPORT**  
880 **ELSE NO\_OPTION**  
881 *Conformance for signal.h: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

882 **29** **IF** *{\_POSIX\_REALTIME\_SIGNALS}* **THEN**  
883 **IF** *PCTS\_timer\_settime* and *PCTS\_timer\_create* **THEN**  
884 **SETUP:** Create a signal so that *si\_code* contains the value *SI\_TIMER*.  
885 **TEST:** The *si\_value* contains the application-specified signal value.  
886 **ELSE NO\_TEST\_SUPPORT**  
887 **ELSE NO\_OPTION**  
888 *Conformance for signal.h: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

889 **30** **IF** *{\_POSIX\_ASYNCHRONOUS\_IO}* **THEN**  
890 **SETUP:** Create a signal so that *si\_code* contains the value *SI\_ASYNCIO*.  
891 **TEST:** The *si\_value* contains the application-specified signal value.  
892 **ELSE NO\_OPTION**  
893 *Conformance for signal.h: PASS, NO\_OPTION*

894 **31** **IF** *{\_POSIX\_MESSAGE\_PASSING}* **THEN**  
895 **SETUP:** Create a signal so that *si\_code* contains the value *SI\_MSGQ*.  
896 **TEST:** The *si\_value* contains the application-specified signal value.  
897 **ELSE NO\_OPTION**  
898 *Conformance for signal.h: PASS, NO\_OPTION*

899 **D\_8** **IF** a PCD.1b documents the following **THEN**  
900 **TEST:** A PCD.1b that documents the *context* parameter to a signal-catching function does so  
901 in §3.3.1.3.  
902 **ELSE NO\_OPTION**  
903 *Conformance for signal.h: PASS, NO\_OPTION*

904 **D\_9** **IF** a PCD.1b documents the following **THEN**  
905 **TEST:** A PCD.1b that documents the behavior of a process after it returns normally from a  
906 signal-catching function for a *SIGFPE*, *SIGHILL*, *SIGSEGV*, or *SIGBUS* signal that was not  
907 generated by the *kill()* function, the *sigqueue()* function, or the *raise()* function as  
908 defined by the C Standard {2} does so in §3.3.1.3.  
909 **ELSE NO\_OPTION**  
910 *Conformance for signal.h: PASS, NO\_OPTION*

911 **32** **IF** *PCTS\_aio\_error* **THEN**

912                   **TEST:**     The function `aiο_error()` is reentrant with respect to signals (that is, applications may  
913                                    invoke them, without restriction, from signal-catching functions).  
914                   **ELSE NO\_OPTION**  
915                   *Conformance for signal.h: PASS, NO\_OPTION*

916    **33**           **IF PCTS\_aiο\_return THEN**  
917                   **TEST:**     The function `aiο_return()` is reentrant with respect to signals (that is, applications  
918                                    may invoke them, without restriction, from signal-catching functions).  
919                   **ELSE NO\_OPTION**  
920                   *Conformance for signal.h: PASS, NO\_OPTION*

921    **34**           **IF PCTS\_aiο\_suspend THEN**  
922                   **TEST:**     The function `aiο_suspend()` is reentrant with respect to signals (that is, applications  
923                                    may invoke them, without restriction, from signal-catching functions).  
924                   **ELSE NO\_OPTION**  
925                   *Conformance for signal.h: PASS, NO\_OPTION*

926    **35**           **IF PCTS\_clock\_gettime THEN**  
927                   **TEST:**     The function `clock_gettime()` is reentrant with respect to signals (that is, applications  
928                                    may invoke them, without restriction, from signal-catching functions).  
929                   **ELSE NO\_OPTION**  
930                   *Conformance for signal.h: PASS, NO\_OPTION*

931    **36**           **IF PCTS\_fdatasync THEN**  
932                   **TEST:**     The function `fdatasync()` is reentrant with respect to signals (that is, applications may  
933                                    invoke them, without restriction, from signal-catching functions).  
934                   **ELSE NO\_OPTION**  
935                   *Conformance for signal.h: PASS, NO\_OPTION*

936    **37**           **IF PCTS\_sem\_post THEN**  
937                   **TEST:**     The function `sem_post()` is reentrant with respect to signals (that is, applications may  
938                                    invoke them, without restriction, from signal-catching functions).  
939                   **ELSE NO\_OPTION**  
940                   *Conformance for signal.h: PASS, NO\_OPTION*

941    **38**           **IF PCTS\_sig\_queue THEN**  
942                   **TEST:**     The function `sig_queue()` is reentrant with respect to signals (that is, applications may  
943                                    invoke them, without restriction, from signal-catching functions).  
944                   **ELSE NO\_OPTION**  
945                   *Conformance for signal.h: PASS, NO\_OPTION*

946    **39**           **IF PCTS\_timer\_getoverrun THEN**  
947                   **TEST:**     The function `timer_getoverrun()` is reentrant with respect to signals (that is,  
948                                    applications may invoke them, without restriction, from signal-catching functions).  
949                   **ELSE NO\_OPTION**  
950                   *Conformance for signal.h: PASS, NO\_OPTION*

951    **40**           **IF PCTS\_timer\_gettime THEN**  
952                   **TEST:**     The function `timer_gettime()` is reentrant with respect to signals (that is, applications  
953                                    may invoke them, without restriction, from signal-catching functions).  
954                   **ELSE NO\_OPTION**  
955                   *Conformance for signal.h: PASS, NO\_OPTION*

956    **41**           **IF PCTS\_timer\_settime THEN**  
957                   **TEST:**     The function `timer_settime()` is reentrant with respect to signals (that is, applications  
958                                    may invoke them, without restriction, from signal-catching functions).  
959                   **ELSE NO\_OPTION**  
960                   *Conformance for signal.h: PASS, NO\_OPTION*



961 **3.3.1.4 Signal Effects on Other Functions**

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

963 NOTE: There are assertions in IEEE Std 2003.1-1992 {4} for this subclause that mention specific functions. There  
964 may need to be additional assertions here that correspond to those assertions in IEEE Std 2003.1-1992 {4} and cover  
965 the new functions of IEEE Std 1003.1b-1993.

966 **3.3.2 Send a Signal to a Process**

967 Function: *kill()*

968 **3.3.2.1 Synopsis**

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

970 **3.3.2.2 Description**

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

972 **3.3.2.3 Returns**

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

974 **3.3.2.4 Errors**

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

976 **3.3.3 Manipulate Signal Sets**

977 Functions: *sigemptyset()*, *sigfillset()*, *sigaddset()*, *sigdelset()*, *sigismember()*

978 **3.3.3.1 Synopsis**

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

980 **3.3.3.2 Description**

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

982 **3.3.3.3 Returns**

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

984 **3.3.3.4 Errors**

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

986 **3.3.4 Examine and Change Signal Action**

987 Function: *sigaction()*

988 **3.3.4.1 Synopsis**

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

990 **3.3.4.2 Description**

991 **1** **SETUP:** Call *sigaction()* with the SA\_SIGINFO flag cleared in the *sa\_flags* field of the *sigaction*  
 992 structure to establish a signal-catching function.  
 993 **TEST:** The *sa\_handler* field identifies the action to be associated with the specified signal.  
 994 **TR:** Test for all catchable signals consistent with implemented options.  
 995 *Conformance for sigaction: PASS*

996 **2** **IF** {\_POSIX\_REALTIME\_SIGNALS} **THEN**  
 997 **SETUP:** Call *sigaction()* with the SA\_SIGINFO flag set in the *sa\_flags* field of the *sigaction*  
 998 structure to establish a signal-catching function.  
 999 **TEST:** The *sa\_sigaction* field specifies a signal-catching function.  
 1000 **TR:** Test for all catchable signals consistent with implemented options.  
 1001 **ELSE NO\_OPTION**  
 1002 *Conformance for sigaction: PASS, NO\_OPTION*

1003 **3** **SETUP:** Call *sigaction()* with the SA\_SIGINFO bit cleared and the *sa\_handler* field specifying a  
 1004 signal-catching function and perform the test. Then call it again with the SA\_SIGINFO bit  
 1005 set and perform the test.  
 1006 **TEST:** The *sa\_mask* field identifies a set of signals that are added to the signal mask of the process  
 1007 before the signal-catching function is invoked.  
 1008 **TR:** Test for all catchable signals consistent with implemented options. Do not test for SIGKILL and  
 1009 SIGSTOP.  
 1010 *Conformance for sigaction: PASS*

1011 **4** **SETUP:** Include the header `<signal.h>`.  
 1012 **TEST:** The following flag bits are defined and can be set in *sa\_flags*:

	<b>Symbolic Constant</b>	<b>Description</b>
1015	SA_NOCLDSTOP	Do not generate SIGCHLD when children stop.
1016	SA_SIGINFO	Invoke the signal-catching function with three arguments instead of one.

1017 *Conformance for sigaction: PASS*

1018 **D\_1 TEST:** The PCD.1b documents the disposition of subsequent occurrences of *sig* when it is already  
 1019 pending and the signal-catching function was established with SA\_SIGINFO not set in *sa\_flags* in  
 1020 §3.3.4.2.  
 1021 *Conformance for sigaction: PASS*

1022 **5** **IF** {POSIX\_REALTIME\_SIGNALS} **THEN**  
 1023 **SETUP:** Set SA\_SIGINFO in *sa\_flags* and establish a signal-catching function by calling  
 1024 *sigaction()*.  
 1025 **TEST:** Subsequent occurrences of *sig* generated by *sigqueue()* or as a result of any signal-  
 1026 generating function that supports the specification of an application-defined value –  
 1027 when *sig* is already pending – is queued in FIFO order until delivered; the signal-  
 1028 catching function is invoked with three arguments; and the application specified  
 1029 value is passed to the signal-catching function as the *si\_value* member of the  
 1030 *siginfo\_t* structure.  
 1031 **TR:** Test for the following functions:

- 1032 1. If {\_POSIX\_REALTIME\_SIGNALS} is defined then *kill()*.
- 1033 2. If *PCTS\_sigqueue* then *sigqueue()*.
- 1034 3. If *PCTS\_lio\_listio* then *lio\_listio()*.
- 1035 4. If *PCTS\_timer\_create* and *PCTS\_timer\_settime* then *timer\_settime()*.

1036 5. If *PCTS\_mq\_open* and *PCTS\_mq\_send* and *PCTS\_mq\_notify* then *mq\_notify()*.

1037 **ELSE NO\_OPTION**

1038 *Conformance for sigaction: PASS, NO\_OPTION*

### 1039 3.3.4.3 Returns

1040 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

### 1041 3.3.4.4 Errors

1042 **6 IF** {POSIX\_REALTIME\_SIGNALS} is not supported **THEN**

1043 **TEST:** A call to *sigaction()* with the SA\_SIGINFO bit flag set in the *sa\_flags* field of the  
1044 *sigaction* structure returns a value of -1 and sets *errno* to [ENOTSUP].

1045 **ELSE NO\_OPTION**

1046 *Conformance for sigaction: PASS, NO\_OPTION*

## 1047 3.3.5 Examine and Change Blocked Signals

1048 Function: *sigprocmask()*

### 1049 3.3.5.1 Synopsis

1050 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

### 1051 3.3.5.2 Description

1052 **D\_1 IF** a PCD.1b documents the following **THEN**

1053 **TEST:** A PCD.1b that documents the result of generating a SIGFPE, SIGILL, SIGSEGV, or SIGBUS  
1054 signal while they are blocked, when the signal was not generated by the *kill()*  
1055 function, the *sigqueue()* function, or the *raise()* function as defined by the C Standard  
1056 {2} does so in §3.3.5.2.

1057 **ELSE NO\_OPTION**

1058 *Conformance for sigprocmask: PASS, NO\_OPTION*

### 1059 3.3.5.3 Returns

1060 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

### 1061 3.3.5.4 Errors

1062 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

## 1063 3.3.6 Examine Pending Signals

1064 Function: *sigpending()*

### 1065 3.3.6.1 Synopsis

1066 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

### 1067 3.3.6.2 Description

1068 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

### 1069 3.3.6.3 Returns

1070 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

1071 **3.3.6.4 Errors**

1072

1073 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

1074 **3.3.7 Wait for a Signal**

1075 Function: *sigsuspend()*

1076 **3.3.7.1 Synopsis**

1077 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

1078 **3.3.7.2 Description**

1079 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

1080 **3.3.7.3 Returns**

1081 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

1082 **3.3.7.4 Errors**

1083 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

1084 **3.3.8 Synchronously Accept a Signal**

1085 Function: *sigwaitinfo()*, *sigtimedwait()*

1086 **3.3.8.1 Synopsis**

1087 **1**

1088 *M\_GA\_stdC\_proto\_decl(int; sigwaitinfo: const sigset\_t \*set, siginfo\_t \*info; signal.h;;)*

1089 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3

1090 *Conformance for sigwaitinfo: PASS[1, 2], NO\_OPTION*

1091 **2**

1092 *M\_GA\_commonC\_int\_result\_decl(sigwaitinfo; signal.h;;)*

1093 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3

1094 *Conformance for sigwaitinfo: PASS[1, 2], NO\_OPTION*

1095 **3**

1096 *M\_GA\_macro\_result\_decl(int; sigwaitinfo; signal.h;;)*

1097 **SEE:** Assertion GA\_macro\_result in §1.3.4

1098 *Conformance for sigwaitinfo: PASS, NO\_OPTION*

1099 **4**

1100 *M\_GA\_macro\_args(sigwaitinfo; signal.h;;)*

1101 **SEE:** Assertion GA\_macro\_args in §2.7.3

1102 *Conformance for sigwaitinfo: PASS, NO\_OPTION*

1103 **5**

1104 *M\_GA\_stdC\_proto\_decl(int; sigtimedwait; , const sigset\_t \*set, siginfo\_t \*info, const struct timespec*  
 1105 *\*timeout; signal.h;;)*

1106 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3

1107 *Conformance for sigtimedwait: PASS[5, 6], NO\_OPTION*

- 1108 **6**  
 1109 *M\_GA\_commonC\_int\_result\_decl(sigtimedwait; , const sigset\_t \*set, siginfo\_t \*info, const struct*  
 1110 *timespec \*timeout; signal.h;;;)*  
 1111 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 1112 *Conformance for sigtimedwait: PASS, NO\_OPTION*
- 1113 **7**  
 1114 *M\_GA\_macro\_result\_decl(int; sigtimedwait; signal.h;;;)*  
 1115 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 1116 *Conformance for sigtimedwait: PASS, NO\_OPTION*
- 1117 **8**  
 1118 *M\_GA\_macro\_args (sigtimedwait; signal.h;;;)*  
 1119 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 1120 *Conformance for sigtimedwait: PASS, NO\_OPTION*
- 1121 **3.3.8.2 Description**
- 1122 **9** **IF** *PCTS\_sigwaitinfo* **THEN**  
 1123 **TEST:** The function *sigwaitinfo()* waits for the pending signal from the set of signals  
 1124 specified by the *set* parameter and returns the selected signal number.  
 1125 **TR :** Test for only one signal at a time by specifying them in *set* then send two signals, one f  
 1126 which is the signal specified in *set*. Test this for signals in the range SIGRTMIN to  
 1127 SIGRTMAX. Then do the same test separately for all non-realtime signals supported by the  
 1128 implementation.  
 1129 **ELSE NO\_OPTION**  
 1130 *Conformance for sigwaitinfo: PASS, NO\_OPTION*
- 1131 **10** **FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
 1132 **IF** *PCTS\_function* **THEN**  
 1133 **TEST:** When there are multiple pending signals in the range SIGRTMIN to SIGRTMAX which  
 1134 have been specified in the *set* argument to *function()* the signal number returned is  
 1135 the lowest numbered one.  
 1136 **NOTE:** This can be tested by creating two processes, one to send signals and one to wait for  
 1137 them by calling *function()*. Block the multiple signals to be tested. Then send those  
 1138 signals. Then wait for signals by calling *function()*.  
 1139 **ELSE NO\_OPTION**  
 1140 *Conformance for sigwaitinfo, sigtimedwait: PASS, NO\_OPTION*
- 1141 **D\_1 FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
 1142 **IF** *PCTS\_function* and a PCD.1 documents the following **THEN**  
 1143 **TEST:** A PCD.1b that documents the selection order between realtime and nonrealtime  
 1144 signals, or between multiple pending nonrealtime signals does so in §3.3.8.2.  
 1145 **ELSE NO\_OPTION**  
 1146 *Conformance for sigwaitinfo, sigtimedwait: PASS, NO\_OPTION*
- 1147 **11** **FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
 1148 **IF** *PCTS\_function* **THEN**  
 1149 **SETUP:** Call *function()* with no signal in *set* pending at the time of the call.  
 1150 **TEST:** The process calling *function()* is suspended until one or more signals in *set* become  
 1151 pending, in which case the selected signal number is returned, or until it is  
 1152 interrupted by an unblocked, caught signal.  
 1153 **TR:** Test for both cases: when no signals are pending and when the call is interrupted by an  
 1154 unblocked, caught signal.  
 1155 **ELSE NO\_OPTION**  
 1156 *Conformance for sigwaitinfo, sigtimedwait: PASS, NO\_OPTION*
- 1157 **12** **FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
 1158 **IF** *PCTS\_function* **THEN**

1159                   **SETUP:** Call *function()* with the *info* argument non-NULL  
1160                   **TEST:** The selected signal number is stored in the *si\_signo* member, and the cause of the  
1161                   signal is stored in the *si\_code* member of the *siginfo\_t* structure pointed to by the *info*  
1162                   argument and the selected signal number is returned.  
1163                   **ELSE NO\_OPTION**  
1164                   Conformance for *sigwaitinfo*, *sigtimedwait*: PASS, NO\_OPTION

1165   **13**           **FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
1166                   **IF** *PCTS\_function* **THEN**  
1167                   **SETUP:** Queue different application-specified values for the signal to be selected by a call to  
1168                   *function()* with a non-NULL value in the *info* argument.  
1169                   **TEST:** The first queued value is dequeued and stored in the *si\_value* member of the *siginfo\_t*  
1170                   structure pointed to by the *info* argument after a call to *function()* with a non-NULL  
1171                   *info* argument and the selected signal number is returned.  
1172                   **TR:** Test that all queued values are retrieved in queued order.  
1173                   **ELSE NO\_OPTION**  
1174                   Conformance for *sigwaitinfo*, *sigtimedwait*: PASS, NO\_OPTION

1175   **14**           **FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
1176                   **IF** *PCTS\_function* **THEN**  
1177                   **TEST:** The system resource used to queue the signal selected by *function()* is released and  
1178                   made available to queue other signals when *sigwaitinfo()* returns with the selected  
1179                   signal number.  
1180                   **ELSE NO\_OPTION**  
1181                   Conformance for *sigwaitinfo*, *sigtimedwait*: PASS, NO\_OPTION

1182   **D\_2** **FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
1183                   **IF** *PCTS\_function* and a PCD.1b documents the following **THEN**  
1184                   **TEST:** A PCD.1b that documents the content of the *si\_value* member of the *info* argument  
1185                   when no value is queued does so in §3.3.8.2.  
1186                   **ELSE NO\_OPTION**  
1187                   Conformance for *sigwaitinfo*, *sigtimedwait*: PASS, NO\_OPTION

1188   **15**           **FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
1189                   **IF** *PCTS\_function* **THEN**  
1190                   **TEST:** After dequeuing all pending signals selected by *function()*, the pending indication for  
1191                   each signal is reset.  
1192                   **ELSE NO\_OPTION**  
1193                   Conformance for *sigwaitinfo*, *sigtimedwait*: PASS, NO\_OPTION

1194   **16**           **IF** *PCTS\_sigtimedwait* **THEN**  
1195                   **SETUP:** Call *sigtimedwait()* when none of the signals specified by the *set* arguments are  
1196                   pending.  
1197                   **TEST:** The function *sigtimedwait()* waits for the time interval specified in the *timespec*  
1198                   structure referenced by *timeout* before returning.  
1199                   **ELSE NO\_OPTION**  
1200                   Conformance for *sigtimedwait*: PASS, NO\_OPTION

1201   **17**           **IF** *PCTS\_sigtimedwait* **THEN**  
1202                   **SETUP:** Call *sigtimedwait()* with the *timespec* structure pointed to by *timeout* being zero-  
1203                   valued and none of the signals specified by *set* pending.  
1204                   **TEST:** The *sigtimedwait()* returns immediately with an error.  
1205                   **ELSE NO\_OPTION**  
1206                   Conformance for *sigtimedwait*: PASS, NO\_OPTION

1207   **D\_3** **IF** *PCTS\_sigtimedwait* and a PCD.1b documents the following **THEN**  
1208                   **TEST:** A PCD.1b that documents the behavior with the *timeout* argument to *sigtimedwait()*  
1209                   is the NULL pointer does so in §3.3.8.2.  
1210                   **ELSE NO\_OPTION**

- 1211 *Conformance for sigtimedwait: PASS, NO\_OPTION*
- 1212 **18 FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
 1213 **IF** *PCTS\_function* **THEN**  
 1214 **TEST:** While *function()* is waiting and a signal occurs that is eligible for delivery (i.e., not  
 1215 blocked by the process signal mask), that signal is handled asynchronously and the  
 1216 wait is interrupted.  
 1217 **ELSE NO\_OPTION**  
 1218 *Conformance for sigwaitinfo, sigtimedwait: PASS, NO\_OPTION*
- 1219 **D\_4 FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
 1220 **IF** not {*\_POSIX\_REALTIME\_SIGNALS*} and a PCD.1b documents the following **THEN**  
 1221 **TEST:** A PCD.1b that documents its support or lack of support for *function()* does so in  
 1222 §3.3.8.2.  
 1223 **ELSE NO\_OPTION**  
 1224 *Conformance for sigwaitinfo, sigtimedwait: PASS, NO\_OPTION*
- 1225 **3.3.8.3 Returns**
- 1226 **R\_1 TEST:** Upon successful completion (that is, one of the signals specified by *set* is pending or is  
 1227 generated) *sigwaitinfo()* and *sigtimedwait()* return the selected signal number.  
 1228 **SEE:** All assertions in §3.3.8.2.
- 1229 **R\_2 TEST:** The functions *sigwaitinfo()* and *sigtimedwait()* return a value of -1 and set *errno* to indicate the  
 1230 error.  
 1231 **SEE:** All assertions in §3.3.8.4.
- 1232 **3.3.8.4 Errors**
- 1233 **19 FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
 1234 **IF** *PCTS\_function* **THEN**  
 1235 **SETUP:** Call *function()* to wait on an unblocked signal that will be caught.  
 1236 **TEST:** The function *function()* returns -1 and sets *errno* to [EINTR] when the wait was  
 1237 interrupted by an unblocked, caught signal.  
 1238 **TR:** Test for each catchable signal consistent with the implemented options.  
 1239 **ELSE NO\_OPTION**  
 1240 *Conformance for sigwaitinfo, sigtimedwait: PASS, NO\_OPTION*
- 1241 **20 FOR:** *PCTS\_sigwaitinfo* and *PCTS\_sigtimedwait*  
 1242 **IF** not *PCTS\_function* **THEN**  
 1243 **TEST:** A call to *function()* returns -1 and sets *errno* to [ENOSYS].  
 1244 **ELSE NO\_OPTION**  
 1245 *Conformance for sigwaitinfo, sigtimedwait: PASS, NO\_OPTION*
- 1246 **21 IF** not *PCTS\_sigtimedwait* **THEN**  
 1247 **TEST:** A call to *sigtimedwait()* returns -1 and sets *errno* to [ENOSYS].  
 1248 **ELSE NO\_OPTION**  
 1249 *Conformance for sigtimedwait: PASS, NO\_OPTION*
- 1250 **22 IF** *PCTS\_sigwaitinfo* and *PCTS\_SIGTIMEDWAIT\_VALUE* **THEN**  
 1251 **TEST:** The *sigtimedwait()* function when called with a timeout argument specifying a  
 1252 *tv\_nsec* value less than zero or greater than or equal to 1000 million, and when no  
 1253 signal is pending in *set* and it is necessary to wait, returns -1 and sets *errno* to  
 1254 [EINVAL].  
 1255 **ELSE NO\_OPTION**  
 1256 *Conformance for sigwaitinfo: PASS, NO\_OPTION*
- 1257 **D\_5 IF** *PCTS\_sigwaitinfo* and *PCTS\_SIGTIMEDWAIT\_VALUE* **THEN**

1258                   **TEST:**     A PCD.1b that documents the detection of a *timeout* argument specifying a *tv\_nsec*  
 1259                                   value less than zero or greater than or equal to 1000 million and the conditions under  
 1260                                   which the error is detected does so in §3.3.8.4.  
 1261                   **ELSE NO\_OPTION**  
 1262                   *Conformance for sigwaitinfo: PASS, NO\_OPTION*

### 1263     **3.3.9 Queue a Signal to a Process**

1264     Function: *sigqueue()*

#### 1265     **3.3.9.1 Synopsis**

1266     **1**  
 1267                   *M\_GA\_stdC\_proto\_decl(int; sigqueue; pid\_t pid, int signo, const union signal value; signal.h;;;)*  
 1268     **SEE:**       Assertion GA\_stC\_proto\_decl in §2.7.3  
 1269                   *Conformance for sigqueue: PASS[1,2], NO\_OPTION*

1270     **2**  
 1271                   *MG\_GA\_commonC\_int\_result\_decl(sigqueue; signal.h;;;)*  
 1272     **SEE:**       Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 1273                   *Conformance for sigqueue: PASS[1,2], NO\_OPTION*

1274     **3**  
 1275                   *MG\_GA\_macro\_result\_decl(int; sigqueue; signal.h;;;)*  
 1276     **SEE:**       Assertion GA\_macro\_result\_decl in §1.3.4  
 1277                   *Conformance for sigqueue: PASS, NO\_OPTION*

1278     **4**  
 1279                   *MG\_GA\_macro\_args(sigqueue; signal.h;;;)*  
 1280     **SEE:**       Assertion GA\_macro\_args in §2.7.3  
                  *Conformance for sigqueue: PASS, NO\_OPTION*

#### 1281     **3.3.9.2 Description**

1282     *sigqueue\_priv()*=  
 1283                   Create two processes, one to send the signal and one to receive it, such that if *PCTS\_GAP\_sigqueue*, the  
 1284                   real or effective user ID of the calling process does not match the real or effective user ID of the process  
 1285                   to which the signal is being sent; or if {*\_POSIX\_SAVED\_IDS*} is the real or effective user ID of the calling  
 1286                   process matches the real or saved set-user-ID of the process to which the signal is being sent; otherwise,  
 1287                   the real or effective user ID of the calling process matches the real or effective user ID of the process  
 1288                   to which the signal is being sent.

1289     **5**       **IF** *PCTS\_sigqueue* **THEN**  
 1290                   **SETUP:**  
 1291                                   *sigqueue\_priv()*  
 1292                   **TEST:**     The *sigqueue()* function causes the signal specified by *signo* to be sent with the value  
 1293                                   specified by *value* to the process specified by *pid*.  
 1294                   **TR:** Test for all catchable signals consistent with implemented options. Also, test for the  
 1295                                   supported combinations of real and effective user IDs as well as for appropriate privilege  
 1296                                   and saved set-user-ID, if they are supported by the implementation.  
 1297                   **ELSE NO\_OPTION**  
 1298                   *Conformance for sigqueue: PASS, NO\_OPTION*



1299     **6**     **IF** *PCTS\_sigqueue* **THEN**  
1300             **IF** {SIGQUEUE\_MAX} <= *PCTS\_SIGQUEUE\_MAX* **THEN**  
1301                 **SETUP:**  
1302                     *sigqueue\_priv()*  
1303                 **TEST:**     The *sigqueue()* function can queue a total of {SIGQUEUE\_MAX} signals to one  
1304                     or more processes.  
1305                 **TR:**     Test for one and two processes.

1306                     Test for all catchable signals consistent with implemented options. Also, test  
1307                     for the supported combinations of real and effective user IDs as well as for  
1308                     appropriate privilege and saved set-user-ID, if they are supported by the  
1309                     implementation.

1310                 **ELSE NO\_TEST\_SUPPORT**  
1311                 **ELSE NO\_OPTION**  
1312                 *Conformance for sigqueue: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1313     **7**     **IF** *PCTS\_sigqueue* **THEN**  
1314             **IF** {SIGQUEUE\_MAX} > *PCTS\_SIGQUEUE\_MAX* **THEN**  
1315                 **SETUP:**  
1316                     *sigqueue\_priv()*  
1317                 **TEST:**     The *sigqueue()* function can queue a total of *PCTS\_SIGQUEUE\_MAX* signals to one  
1318                     or more processes.  
1319                 **TR:**     Test for one and two processes.

1320                     Test for all catchable signals consistent with implemented options. Also, test  
1321                     for the supported combinations of real and effective user IDs as well as for  
1322                     appropriate privilege and saved set-user-ID, if they are supported by the  
1323                     implementation.

1324                 **ELSE NO\_TEST\_SUPPORT**  
1325                 **ELSE NO\_OPTION**  
1326                 *Conformance for sigqueue: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1327     **8**     **IF** *PCTS\_sigqueue* **THEN**  
1328             **SETUP:**  
1329                 *sigqueue\_priv()*  
1330             **TEST:**     If the *signo* argument is *sigqueue()* is zero (the null signal), error checking is  
1331                     performed but no signal is actually sent.  
1332             **ELSE NO\_OPTION**  
1333             *Conformance for sigqueue: PASS, NO\_OPTION*

1334     **9**     **IF** *PCTS\_sigqueue* **THEN**  
1335             **SETUP:**  
1336                 *sigqueue\_priv()*  
1337                 Block the signal specified in *signo* for the receiving process and clear the SA\_SIGINFO  
1338                 flag for it.  
1339             **TEST:**     A call to *sigqueue()* returns immediately.  
1340             **TR:**     Test for all catchable signals consistent with implemented options. Also, test for the  
1341                     supported combinations of real and effective user IDs as well as for appropriate privilege  
1342                     and saved set-user-ID, if they are supported by the implementation.  
1343             **ELSE NO\_OPTION**  
1344             *Conformance for sigqueue: PASS, NO\_OPTION*

1345     **10**    **IF** *PCTS\_sigqueue* **THEN**  
1346             **SETUP:**  
1347                 *sigqueue\_priv()*  
1348                 Block the signal specified in *signo* for the receiving process and set the SA\_SIGINFO  
1349                 flag for it.  
1350             **TEST:**     A call to *sigqueue()* returns immediately, and if the resources are available to queue  
1351                     the signal, the signal is left queued and pending.

1352                   **TR:** Test for all catchable signals consistent with the implemented options. Also, test for the  
 1353                   supported combinations of real and effective user IDs as well as for appropriate privilege  
 1354                   and saved set-user-ID, if they are supported by the implementation.  
 1355                   **ELSE NO\_OPTION**  
 1356                   *Conformance for sigqueue: PASS, NO\_OPTION*

1357   **11**           **IF** *PCTS\_sigqueue* **THEN**  
 1358                   **SETUP:**  
 1359                               *sigqueue\_priv()*  
 1360                               Do not set SA\_SIGINFO for *signo*  
 1361                   **TEST:**     The *signo* is sent at least once to the receiving process.  
 1362                   **TR:** Test for all catchable signals consistent with the implemented options. Also, test for the  
 1363                   supported combinations of real and effective user IDs as well as for appropriate privilege  
 1364                   and saved set-user-ID, if they are supported by the implementation.  
 1365                   **ELSE NO\_OPTION**  
 1366                   *Conformance for sigqueue: PASS, NO\_OPTION*

1367   **D\_1** **IF** *PCTS\_sigqueue* and a PCD.1b documents the following **THEN**  
 1368                   **TEST:**     A PCD.1b that documents whether or not the *value* parameter is sent to the receiving  
 1369                   process as a result of calling the *sigqueue()* function when SA\_SIGINFO is not set for  
 1370                   *signo* and the conditions under which the *value* is sent does so in §3.3.9.2.  
 1371                   **ELSE NO\_OPTION**  
 1372                   *Conformance for sigqueue: PASS, NO\_OPTION*

1373   **12**           **IF** *PCTS\_sigqueue* **THEN**  
 1374                   **SETUP:**  
 1375                               Set *pid* so that it causes *signo* to be generated for the sending process, and do not  
 1376                   block *signo*.  
 1377                   **TEST:**     Either *signo* or at least the pending, unblocked signal with the lowest number shall  
 1378                   be delivered to the sending process before the *sigqueue()* function returns.  
 1379                   **TR:** Test for all catchable signals consistent with the implemented options. Also, test for the  
 1380                   supported combinations of real and effective user IDs as well as for appropriate privilege  
 1381                   and saved set-user-ID, if they are supported by the implementation.  
 1382                   **ELSE NO\_OPTION**  
 1383                   *Conformance for sigqueue: PASS, NO\_OPTION*  
 1384

1385   **D-2**          **IF** not { *\_POSIX\_REALTIME\_SIGNALS* } and a PCD.1b documents the following  
 1386                   **THEN**  
 1387                   **TEST:**     A PCD.1b that documents its support or lack of support for *sigqueue()* does so in  
 1388                   §3.3.8.2.  
 1389                   **ELSE NO\_OPTION**  
 1390                   *Conformance for sigqueue: PASS, NO\_OPTION*

### 1391   **3.3.9.3 Returns**

1392   **R-1**          **IF** *PCTS\_sigqueue* **THEN**  
 1393                   **TEST:**     Upon successful completion of a call to *sigqueue()*, the specified signal has been  
 1394                   queued, and the function returns a value of zero.  
 1395                   **ELSE NO\_OPTION**  
 1396                   **SEE:**     Assertions in §3.3.9.2.  
 1397  
 1398   **R-2**          **IF** *PCTS\_sigqueue* **THEN**  
 1399                   **TEST:**     An unsuccessful call to *sigqueue()* returns a value of -1 and sets *errno* to indicate the  
 1400                   error.  
 1401                   **ELSE NO\_OPTION**  
 1402                   **SEE:**     Assertions in §3.3.9.4.

1403 **3.3.9.4 Errors**

- 1404 **13** **IF** *PCTS\_sigqueue* **THEN**  
 1405 **TEST:** A call to *sigqueue()* returns -1 and sets *errno* to [EAGAIN] when no resources are  
 1406 available to queue the signal.  
 1407 **NOTE:** There is no known portable test method for this assertion.  
 1408 **ELSE NO\_OPTION**  
 1409 *Conformance for sigqueue: PASS, NO\_TEST, NO\_OPTION*
- 1410 **14** **IF** *PCTS\_sigqueue* **THEN**  
 1411 **TEST:** A call to *sigqueue()* returns -1 and sets *errno* to [EAGAIN] when the process has  
 1412 already queued {SIGQUEUE\_MAX} signals that are still pending at the receiver(s).  
 1413 **NOTE:** There is no known portable test method for this assertion.  
 1414 **TR:** Test limit for signals sent to one process and to two processes.  
 1415 **ELSE NO\_OPTION**  
 1416 *Conformance for sigqueue: PASS, NO\_OPTION*
- 1417 **15** **IF** *PCTS\_sigqueue* **THEN**  
 1418 **TEST:** A call to *sigqueue()* returns -1 and sets *errno* to [EAGAIN] when a systemwide  
 1419 resource limit has been exceeded.  
 1420 **NOTE:** There is no known portable test method for this assertion.  
 1421 **ELSE NO\_OPTION**  
 1422 *Conformance for sigqueue: PASS, NO\_TEST, NO\_OPTION*
- 1423 **16** **IF** *PCTS\_sigqueue* **THEN**  
 1424 **IF** *PCTS\_INVALID\_SIGNAL* or *PCTS\_UNSUPPORTED\_SIGNAL* **THEN**  
 1425 **TEST:** A call to *sigqueue()* returns -1 and sets *errno* to [EINVAL] when the value of the  
 1426 *signo* argument is an invalid or unsupported signal number.  
 1427 **TR:** Test for both invalid and unsupported signals, if each exists.  
 1428 **ELSE NO\_TEST\_SUPPORT**  
 1429 **ELSE NO\_OPTION**  
 1430 *Conformance for sigqueue: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 1431 **17** **IF** not *PCTS\_sigqueue* **THEN**  
 1432 **TEST:** A call to *sigqueue()* returns -1 and sets *errno* to [ENOSYS] when the function  
 1433 *sigqueue()* is not supported by this implementation.  
 1434 **ELSE NO\_OPTION**  
 1435 *Conformance for sigqueue: PASS, NO\_OPTION*
- 1436 **18** **IF** *PCTS\_sigqueue* **THEN**  
 1437 **IF** *PCTS\_RAP\_sigqueue* **THEN**  
 1438 **TEST:** A call to *sigqueue()* returns -1 and sets *errno* to [EPERM] when the process does  
 1439 not have the appropriate privilege to send the signal to the receiving process.  
 1440 **TR:** Use *sigqueue\_priv()*.  
 1441 **ELSE NO\_TEST\_SUPPORT**  
 1442 **ELSE NO\_OPTION**  
 1443 *Conformance for sigqueue: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 1444 **19** **IF** *PCTS\_sigqueue* **THEN**  
 1445 **TEST:** A call to *sigqueue()* returns -1 and sets *errno* to [ESDRCH] when the process *pid* does  
 1446 not exist.  
 1447 **ELSE NO\_OPTION**  
 1448 *Conformance for sigqueue: PASS, NO\_OPTION*

1449 **3.4 Timer Operations**

1450 There are no requirements for conforming implementations in this subclause.

1451     **3.4.1     Schedule Alarm**

1452     Function: *alarm()*

1453     **3.4.1.1 Synopsis**

1454     There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b (3) assertions.

1455     **3.4.1.2 Description**

1456     There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b (3) assertions.

1457     **3.4.1.3 Returns**

1458     There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b (3) assertions.

1459     **3.4.1.4 Errors**

1460     There are no requirements for conforming implementations in this subclause.

1461     **3.4.2 Suspend Process Execution**

1462     Function: *pause()*

1463     **3.4.2.1 Synopsis**

1464     There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b (3) assertions.

1465     **3.4.2.2 Description**

1466     There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b (3) assertions.

1467     **3.4.2.3 Returns**

1468     There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b (3) assertions.

1469     **3.4.2.4 Errors**

1470     There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b (3) assertions.

1471     **3.4.3 Delay Process Execution**

1472     Function: *sleep()*

1473     **3.4.3.1 Synopsis**

1474     There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b (3) assertions.

1475     **3.4.3.2 Description**

1476     There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b (3) assertions.

1477     **3.4.3.3 Returns**

1478     There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b (3) assertions.

1479 **3.4.3.4 Errors**

1480 There are no requirements for conforming implementations in this subclause.

## Section 4: Process Environment

180 There are no POSIX.1b {3} assertions in Section 4 except for subclause 4.8.1.2.

### 181 4.8 Configurable System Variables

#### 182 4.8.1 Get Configurable System Variables

183 Function: *sysconf()*

##### 184 4.8.1.1 Synopsis

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

##### 186 4.8.1.2 Description

187 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}.

188 (IEEE Std 2003.1-1992 {4} 04

189 *UNUSED*

190 **1**       **SETUP:** Include the header `<unistd.h>`.  
 191       **TEST:**    The symbolic constants in the *name* Value column of Table 4-2 are defined and have  
 192                   different values.  
 193       **NOTE:**    The assertion is tested once for each constant specified in the FOR clause. The assertion is  
 194                   to be read by substituting `CONSTANT` with the current constant specified in the FOR clause.  
 195                   The name of the compound constant also is to be substituted for each occurrence in the  
 196                   constructs `_SC_CONSTANT` and `_POSIX_CONSTANT`.  
 197                   *Conformance for sysconf: PASS*

198 **2**       **FOR:**     {AIO\_LISTIO\_MAX}, {AIO\_MAX}, {AIO\_PRIO\_DELTA\_MAX}, {DELAYTIMER\_MAX},  
 199                   {MQ\_OPEN\_MAX}, {MQ\_PRIO\_MAX}, {PAGESIZE}, {RTSIG\_MAX}, {SEM\_NSEMS\_MAX},  
 200                   {SEM\_VALUE\_MAX}, {SIGQUEUE\_MAX}, {TIMER\_MAX},  
 201       **IF** `CONSTANT` is defined in `<limits.h>` **THEN**  
 202           **SETUP:**    Include the header `<limits.h>`  
 203           **TEST:**     A call `sysconf(_SC_CONSTANT)` either returns -1 without changing the value of *errno*  
 204                       or returns a value greater than or equal to `{CONSTANT}`.  
 205           **NOTE:**     The assertion is tested once for each constant specified in the FOR clause. The  
 206                       assertion is to be read by substituting `CONSTANT` with the current constant specified  
 207                       in the FOR clause. The name of the compound constant also is to be substituted for  
 208                       each occurrence in the constructs `_SC_CONSTANT` and `_POSIX_CONSTANT`.  
 209       **ELSE NO\_TEST\_SUPPORT**  
 210           *Conformance for sysconf: PASS, NO\_TEST\_SUPPORT*

```

211 3  FOR:      {AIO_LISTIO_MAX}, {AIO_MAX}, {AIO_PRIO_DELTA_MAX}, {DELAYTIMER_MAX},
212      {MQ_OPEN_MAX}, {MQ_PRIO_MAX}, {PAGESIZE}, {RTSIG_MAX}, {SEM_NSEMS_MAX},
213      {SEM_VALUE_MAX}, {SIGQUEUE_MAX}, {TIMER_MAX},
214  IF CONSTANT is not defined in <limits.h> THEN
215      SETUP:   Include the header <limits.h>
216      TEST:    A call sysconf(_SC_CONSTANT) either returns -1 without changing the value of errno
217      or returns a value greater than or equal to {_POSIX_CONSTANT}.
218      NOTE:   The assertion is tested once for each constant specified in the FOR clause. The
219      assertion is to be read by substituting CONSTANT with the current constant specified
220      in the FOR clause. The name of the compound constant also is to be substituted for
221      each occurrence in the constructs _SC_CONSTANT and _POSIX_CONSTANT.
222  ELSE NO_TEST_SUPPORT
223  Conformance for sysconf: PASS, NO_TEST_SUPPORT

224 4  FOR:      {_POSIX_ASYNCHRONOUS_IO}, { _POSIX_FSYNC},
225      {_POSIX_MAPPED_FILES}, { _POSIX_MEMLOCK},
226      {_POSIX_MEMLOCK_RANGE}, { _POSIX_MEMORY_PROTECTION},
227      {_POSIX_MESSAGE_PASSING}, { _POSIX_PRIORITIZED_IO},
228      {_POSIX_PRIORITY_SCHEDULING}, { _POSIX_REALTIME_SIGNALS},
229      {_POSIX_SEMAPHORES}, { _POSIX_SHARED_MEMORY_OBJECTS},
230      {POSIX_SYNCHRONIZED_IO}, { _POSIX_TIMERS},
231  IF CONSTANT is defined in <unistd.h> THEN
232      SETUP:   Include the header <unistd.h>
233      TEST:    A call sysconf(_SC_CONSTANT) returns a value other than -1.
234      NOTE:   The assertion is tested once for each constant specified in the FOR clause. The
235      assertion is to be read by substituting CONSTANT with the current constant specified
236      in the FOR clause. The name of the compound constant also is to be substituted for
237      each occurrence in the constructs _SC_CONSTANT and _POSIX_CONSTANT.
238  ELSE NO_TEST_SUPPORT
239  Conformance for sysconf: PASS, NO_TEST_SUPPORT

```

240

**Table 4-2 – Configurable System Variables**

241

	<b>Variable</b>	<b>name Value</b>
242	{AIO_LISTIO_MAX}	{_SC_AIO_LISTIO_MAX}
243	{AIO_MAX}	{_SC_AIO_MAX}
244	{AIO_PRIO_DELTA_MAX}	{_SC_AIO_PRIO_DELTA_MAX}
245	{ART_MAX}	{_SC_ARG_MAX}
246	{CHILD_MAX}	{_SC_CHILD_MAX}
247	clock ticks/second	{_SC_CLK_TCK}
248	{DELAYTIMER_MAX}	{_SC_DELAYTIMER_MAX}
249	{MQ_OPEN_MAX}	{_SC_MQ_OPEN_MAX}
250	{MQ_PRIO_MAX}	{_SC_MQ_PRIO_MAX}
251	{NGROUPS_MAX}	{_SC_NGROUPS_MAX}
252	{OPEN_MAX}	{_SC_OPEN_MAX}
253	{PAGESIZE}	{_SC_PAGESIZE}
254	{RTSIG_MAX}	{_SC_RTSIG_MAX}
255	{SEM_NSEMS_MAX}	{_SC_SEM_NSEMS_MAX}
256	{SEM_VALUE_MAX}	{_SC_SEM_VALUE_MAX}
257	{SIGQUEUE_MAX}	{_SC_SIGQUEUE_MAX}
258	{STREAM_MAX}	{_SC_STREAM_MAX}
259	{TIMER_MAX}	{_SC_TIMER_MAX}
260	{TZNAME__MAX}	{_SC_TZNAME_MAX}
261	{_POSIX_ASYNCHRONOUS_IO}	{_SC_ASYNCHRONOUS_IO}
262	{_POSIX_FSYNC}	{_SC_FSYNC}
263	{_POSIX_JOB_CONTROL}	{_SC_JOB_CONTROL}
264	{_POSIX_MAPPED_FILES}	{_SC_MAPPED_FILES}
265	{_POSIX_MEMLOCK}	{_SC_MEMLOCK}
266	{_POSIX_MEMLOCK_RANGE}	{_SC_MEMLOCK_RANGE}
267	{_POSIX_MEMORY_PROTECTION}	{_SC_MEMORY_PROTECTION}
268	{_POSIX_MESSAGE_PASSING}	{_SC_MESSAGE_PASSING}
269	{_POSIX_PRIORITIZED_IO}	{_SC_PRIORITIZED_IO}
270	{_POSIX_PRIORITY_SCHEDULING}	{_SC_PRIORITY_SCHEDULING}
271	{_POSIX_REALTIME_SIGNALS}	{_SC_REALTIME_SIGNALS}
272	{_POSIX_SAVED_IDS}	{_SC_SAVED_IDS}
273	{_POSIX_SEMAPHORES}	{_SC_SEMAPHORES}
274	{_POSIX_SHARED_MEMORY_OBJECTS}	{_SC_SHARED_MEMORY_OBJECTS}
275	{_POSIX_SYNCHRONIZED_IO}	{_SC_SYNCHRONIZED_IO}
276	{_POSIX_TIMERS}	{_SC_TIMERS}
277	{_POSIX_VERSION}	{_SC_VERSION}

278

**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},

285

**IF** CONSTANT is **not** defined in <unistd.h> **THEN**

286

**SETUP:** Include the header <unistd.h>

287

**TEST:** A call `sysconf(_SC_CONSTANT)` returns -1 without changing the value of `errno`.

288

**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`.

289

290

291



292                   **ELSE NO\_TEST\_SUPPORT**  
293                   *Conformance for sysconf: PASS, NO\_TEST\_SUPPORT*

294    **4.8.1.3 Returns**

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

296    **4.8.1.4 Errors**

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

298    **4.8.1.5 Special Symbol {CLK\_TCK}**

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

## Section 5: Files and Directories

180 There are no requirements for conforming implementations in this subclause.

### 181 **5.1 Directories**

#### 182 **5.1.1 Format of Directory Entries**

183 There are no requirements for conforming implementations in this subclause.

#### 184 **5.1.2 Directory Operations**

185 Functions: *opendir()*, *readdir()*, *rewinddir()*, *closedir()*

##### 186 **5.1.2.1 Synopsis**

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

##### 188 **5.1.2.2 Description**

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

##### 190 **5.1.2.3 Returns**

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

##### 192 **5.1.2.4 Errors**

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

194

### 195 **5.2 Working Directory**

#### 196 **5.2.1 Change Current Working Directory**

197 Function: *chdir()*

##### 198 **5.2.1.1 Synopsis**

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

##### 200 **5.2.1.2 Description**

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

202 **5.2.1.3 Returns**

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

204 **5.2.1.4 Errors**

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

206

207 **5.2.2 Get Working Directory Pathname**208 Function: *getcwd()*209 **5.2.2.1 Synopsis**

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

211 **5.2.2.2 Description**

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

213 **5.2.2.3 Returns**

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

215 **5.2.2.4 Errors**

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

217 **5.3 General File Creation**218 **5.3.1 Open a File**219 Function: *open()*220 **5.3.1.1 Synopsis**

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

222 **5.3.1.2 Description**

223 NOTE: The General Assertion *GA\_syncIODataIntegrityWrite* is defined in §2.2.2.119. It is tested in *write()*,  
 224 *aio\_write()*, and *lio\_listio()* instead of in *open()* because they are more appropriate places to test it. The General  
 225 Assertion *GA\_syncIODataIntegrityRead* is defined in §2.2.2.119. It is tested in *read()*, *aio\_read()*, and *lio\_listio()*  
 226 instead of in *open()* because they are more appropriate places to test it. The General Assertion  
 227 *GA\_syncIOFileIntegrityRead* is defined in §2.2.1.120. It is tested in *read()*, *aio\_read()*, and *lio-listio()* instead of in  
 228 *open()* because they are more appropriate places to test it. The General Assertion *GA\_syncIODataIntegrityWrite* is  
 229 defined in §2.2.2.120. It is tested in *write()*, *aio\_write()*, and *listio()* instead of in *open()* because they are more  
 230 appropriate places to test it.

231 Also, the *O\_SYNC*, *O\_RSYNC*, and *O\_SYNC* constants are tested for existence and value in §6.5.1.1.232 **GA\_syncOpenWrite**233 **FOR:** *write()*, *aio\_write()*, and *lio-listio()*234 **IF** {*\_POSIX\_SYNCHRONIZED\_IO*} **THEN**235 **SETUP:** Open a file by calling *open()* with both *O\_SYNC* and *O\_DSYNC* set in the *oflag*  
 236 parameter.237 **TEST:** The file behaves as if only the *O\_SYNC* flag was set.238 **TR:** Test for regular files.

239                   **NOTE:** There is no known portable test method for this assertion.  
 240                   **ELSE NO\_OPTION**  
 241                   *Conformance for open: PASS, NO\_TEST\_NO\_OPTION*

### 242   **5.3.1.3 Returns**

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

### 244   **5.3.1.4 Errors**

245   **1**           **IF** {\_POSIX\_SYNCHRONIZED\_IO} **THEN**  
 246                   **IF** PCTS\_NO\_SYNC\_IO\_FILE **THEN**  
 247                         **SETUP:** Call *open()*( *path*, *oflag* / *oflag1*).  
 248                         **TEST:** The *open()* returns -1 and sets *errno* to [EINVAL].  
 249                         **NOTE:** This implementation does not support synchronized I/O for this file.  
 250                         **ELSE NO\_TEST\_SUPPORT**  
 251                   **ELSE NO\_OPTION**  
 252                   *Conformance for open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

### 253   **5.3.1.5 Errors**

254   **2**           **IF** {\_POSIX\_SYNCHRONIZED\_IO} **THEN**  
 255                   **IF** PCTS\_NO\_SYNC\_IO\_FILE **THEN**  
 256                         **SETUP:** Call *open()*( *path*, *oflag* / *oflag1*).  
 257                         **TEST:** The *open()* returns -1 and sets *errno* to [EINVAL].  
 258                         **NOTE:** This implementation does not support synchronized I/O for this file.  
 259                         **ELSE NO\_TEST\_SUPPORT**  
 260                   **ELSE NO\_OPTION**  
 261                   *Conformance for open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

## 262   **5.3.2 Create New File or Rewrite an Existing One**

263   Function: *creat()*

### 264   **5.3.2.1 Synopsis**

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

### 266   **5.3.2.2 Description**

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

268

## 269   **5.3.3 Set File Creation Mask**

270   Function: *umask()*

### 271   **5.3.3.1 Synopsis**

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

### 273   **5.3.3.2 Description**

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

### 275   **5.3.3.3 Returns**

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

277 **5.3.3.4 Errors**

278 There are no requirements for conforming implementations in this subclause.

279 **5.3.4 Link to a File**

280 Function: *link()*

281 **5.3.4.1 Synopsis**

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

283 **5.3.4.2 Description**

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

285 **5.3.4.3 Returns**

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

287 **5.3.4.4 Errors**

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

289 **5.4 Special File Creation**

290 **5.4.1 Make a Directory**

291 Function: *mkdir()*

292 **5.4.1.1 Synopsis**

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

294 **5.4.1.2 Description**

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

296 **5.4.1.3 Returns**

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

298 **5.4.1.4 Errors**

299

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

301 **5.4.2 Make a FIFO Special File**

302 Function: *mkfifo()*

303 **5.4.2.1 Synopsis**

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

305 **5.4.2.2 Description**

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

### 307 **5.4.2.3 Returns**

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

### 309 **5.4.2.4 Errors**

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

311

## 312 **5.5 File Removal**

### 313 **5.5.1 Remove Directory Entries**

314 Function: *unlink()*

#### 315 **5.5.1.1 Synopsis**

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

#### 317 **5.5.1.2 Description**

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

#### 319 **5.5.1.3 Returns**

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

#### 321 **5.5.1.4 Errors**

322

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

### 324 **5.5.2 Remove a Directory**

325 Function: *rmdir()*

#### 326 **5.5.2.1 Synopsis**

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

#### 328 **5.5.2.2 Description**

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

#### 330 **5.5.2.3 Returns**

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

#### 332 **5.5.2.4 Errors**

333

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

### 335 **5.5.3 Rename a File**

336 Function: *rename()*

337 **5.5.3.1 Synopsis**

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

339 **5.5.3.2 Description**

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

341 **5.5.3.3 Returns**

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

343 **5.5.3.4 Errors**

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

345 **5.6 File Characteristics**346 **5.6.1 File Characteristics: Header and Data Structure**347 `<sys/stat.h>`

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

349 **5.6.1.1 `<sys/stat.h>` File Types**350 **D\_1** If a PCD.1b documents the following **THEN**351 **TEST:** A PCD.1b that documents whether message queues, semaphores, or shared memory  
352 objects are implemented as distinct file types does so in §5.6.1.1.353 **ELSE NO\_OPTION**354 *Conformance for stat.h: PASS, NO\_OPTION*355 **1** **SETUP:** Include the header `<sys/stat.h>`.356 **TEST:** The macros `S_TYPEISMQ()`, `S_TYPEISSEM()`, and `S_TYPEISSHM()` are defined.357 *Conformance for stat.h: PASS*358 **2** **IF** `PCTS_MQ_AS_FILE_TYPE` **THEN**359 **SETUP:** Include the header `<sys/stat.h>`. Also, create a message queue and put its  
360 information into the `stat` structure referenced by `buf` parameter.361 **TEST:** The macro call `S_TYPEISMQ(buf)` evaluates to a nonzero value.362 **ELSE NO\_TEST\_SUPPORT**363 *Conformance for stat.h: PASS, NO\_TEST\_SUPPORT*364 **3** **IF** `PCTS_MQ_AS_FILE_TYPE` **THEN**365 **SETUP:** Include the header `<sys/stat.h>`. Do not put the information associated with  
366 a message queue type file into the `stat` structure referenced by `buf` parameter.367 **TEST:** The macro call `S_TYPEISMQ(buf)` evaluates to a zero value.368 **ELSE NO\_TEST\_SUPPORT**369 *Conformance for stat.h: PASS, NO\_TEST\_SUPPORT*

370 **4** **IF** Not *PCTS\_MQ\_AS\_FILE\_TYPE* **THEN**  
371 **SETUP:** Include the header `<sys/stat.h>`. Do not put the information associated with  
372 a message queue type file into the *stat* structure referenced by *buf* parameter.  
373 **TEST:** The macro call *S\_TYPEISMQ(buf)* evaluates to a zero value.  
374 **ELSE NO\_TEST\_SUPPORT**  
375 *Conformance for stat.h: PASS, NO\_TEST\_SUPPORT*

376 **5** **IF** *PCTS\_SEM\_IS\_FD* **THEN**  
377 **SETUP:** Include the header `<sys/stat.h>`. Also, create a semaphore and put its  
378 information into the *stat* structure referenced by *buf* parameter.  
379 **TEST:** The macro call *S\_TYPEISSEM(buf)* evaluates to a nonzero value.  
380 **ELSE NO\_TEST\_SUPPORT**  
381 *Conformance for stat.h: PASS, NO\_TEST\_SUPPORT*

382 **6** **IF** *PCTS\_SEM\_IS\_FD* **THEN**  
383 **SETUP:** Include the header `<sys/stat.h>`. Do not put the information associated with  
384 a semaphore type file into the *stat* structure referenced by *buf* parameter.  
385 **TEST:** The macro call *S\_TYPEISSEM(buf)* evaluates to a zero value.  
386 **ELSE NO\_TEST\_SUPPORT**  
387 *Conformance for stat.h: PASS, NO\_TEST\_SUPPORT*

388 **7** **IF** *PCTS\_SEM\_IS\_FD* **THEN**  
389 **SETUP:** Include the header `<sys/stat.h>`. Do not put the information associated with  
390 a semaphore type file into the *stat* structure referenced by *buf* parameter.  
391 **TEST:** The macro call *S\_TYPEISSEM(buf)* evaluates to a zero value.  
392 **ELSE NO\_TEST\_SUPPORT**  
393 *Conformance for stat.h: PASS, NO\_TEST\_SUPPORT*

394 **8** **IF** *PCTS\_SHM\_AS\_FILE\_TYPE* **THEN**  
395 **SETUP:** Include the header `<sys/stat.h>`. Also, create a shared memory object and put  
396 its information into the *stat* structure referenced by *buf* parameter.  
397 **TEST:** The macro call *S\_TYPEISSHM(buf)* evaluates to a nonzero value.  
398 **ELSE NO\_TEST\_SUPPORT**  
399 *Conformance for stat.h: PASS, NO\_TEST\_SUPPORT*

400 **9** **IF** *PCTS\_SHM\_AS\_FILE\_TYPE* **THEN**  
401 **SETUP:** Include the header `<sys/stat.h>`. Do not put the information associated with  
402 a shared memory object type file into the *stat* structure referenced by *buf* parameter.  
403 **TEST:** The macro call *S\_TYPEISSHM(buf)* evaluates to a zero value.  
404 **ELSE NO\_TEST\_SUPPORT**  
405 *Conformance for stat.h: PASS, NO\_TEST\_SUPPORT*

406 **10** **IF** *PCTS\_SHM\_AS\_FILE\_TYPE* **THEN**  
407 **SETUP:** Include the header `<sys/stat.h>`. Do not put the information associated with  
408 a shared memory object type file into the *stat* structure referenced by *buf* parameter.  
409 **TEST:** The macro call *S\_TYPEISSHM(buf)* evaluates to a nonzero value.  
410 **ELSE NO\_TEST\_SUPPORT**  
411 *Conformance for stat.h: PASS, NO\_TEST\_SUPPORT*

#### 412 **5.6.1.2** `<sys/stat.h>` File Modes

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

#### 414 **5.6.1.3** `<sys/stat.h>` Time Entries

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



416 **5.6.2 Get File Status**417 Functions: *stat()*, *fstat()*418 **5.6.2.1 Synopsis**

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

420 **5.6.2.2 Description**421 **1** **SETUP:** Open a shared memory object using *shm\_open()*. Call *fstat()* with the *fildev* parameter  
422 referring to that shared memory object.423 **TEST:** The *stat* structure pointed to by the *buf* argument has the *S\_IRUSR*, *S\_IWUSR*, *S\_IRGRP*,  
424 *S\_IWGRP*, *S\_IROTH*, and *S\_IWOTH* file permission bits updated correctly.425 *Conformance for fstat: PASS*426 **5.6.2.3 Returns**

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

428 **5.6.2.4 Errors**

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

430 **5.6.3 Check File Accessibility**431 Function: *access()*432 **5.6.3.1 Synopsis**

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

434 **5.6.3.2 Description**

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

436 **5.6.3.3 Returns**

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

438 **5.6.3.4 Errors**

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

440 **5.6.4 Change File Modes**441 Function: *chmod()*, *fchmod()*442 **5.6.4.1 Synopsis**443 **1**444 *M\_GA\_stdC\_proto\_decl(int;fchmod;,intfildev, mode\_t mode; sys/stat.h;;; )*445 **SEE:** Assertion *GA\_stdC\_proto\_decl* in §2.7.3446 *Conformance for fchmod: PASS[1, 2], NO\_OPTION*447 **2**

448 *M\_GA\_commonC\_int\_result\_decl(fchmod; sys/stat.h;; )*  
 449 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 450 *Conformance for fchmod: PASS[1, 2], NO\_OPTION*

451 **3**  
 452 *M\_GA\_macro\_result\_decl(int;fchmod; sys/stat.h;; )*  
 453 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 454 *Conformance for fchmod: PASS, NO\_OPTION*

455 **4**  
 456 *M\_GA\_macro\_args(fchmod; sys/stat.h;; )*  
 457 **SEE:** Assertion GA\_macro\_macro\_args in §2.7.3  
 458 *Conformance for fchmod: PASS, NO\_OPTION*

459 **5.6.4.2 Description**

460 **5** **IF** *PCTSfchmod* **THEN**  
 461 **SETUP:** Open a file where the effective user ID of the calling process matches the file owner.  
 462 **TEST:** A call *fchmod(fildes, mode)* sets the file permission bits from the corresponding bits  
 463 in the *mode* argument.  
 464 **TR:** For regular files: test all file permission bits.  
  
 465 For shared memory objects: test only the S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP,  
 466 S\_IROTH, and S\_IWOTH file permission bits.  
 467 **ELSE NO\_OPTION**  
 468 *Conformance for fchmod: PASS, NO\_OPTION*

469 **6** **IF** *PCTS\_fchmod* **THEN**  
 470 **IF** *PCTS\_GAP\_MODES\_fchmod* **THEN**  
 471 **SETUP:** Open a file where the effective user ID of the calling process does not match the  
 472 file owner and acquire the appropriate privilege to change the file permission  
 473 bits of the file using *fchmod()*.  
 474 **TEST:** A call *fchmod(fildes, mode)* sets the file permission bits from the corresponding  
 475 bits in the *mode* argument.  
 476 **TR :** For regular files: test all file permission bits.  
  
 477 For shared memory objects: test only the S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP,  
 478 S\_IROTH, and S\_IWOTH file permission bits.  
 479 **ELSE NO\_TEST\_SUPPORT**  
 480 **ELSE NO\_OPTION**  
 481 *Conformance for fchmod: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

482 **7** **IF** *PCTS\_fchmod* and *PCTS\_CHMOD\_SUID* **THEN**  
 483 **SETUP:** Open a file where the effective user ID of the calling process matches the file owner.  
 484 **TEST:** A call *fchmod(fildes, mode)* sets the S\_ISUID bit from the corresponding bit in the  
 485 *mode* argument.  
 486 **TR:** For regular files: test all file permission bits.  
  
 487 For shared memory objects: test only the S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP,  
 488 S\_IROTH, and S\_IWOTH file permission bits.  
 489 **ELSE NO\_OPTION**  
 490 *Conformance for fchmod: PASS, NO\_OPTION*

491 **8** **IF** *PCTS\_fchmod* and *PCTS\_CHMOD\_SUID* **THEN**  
 492 **IF** *PCTS\_GAP\_SUID\_fchmod* **THEN**  
 493 **SETUP:** Open a file where the effective user ID of the calling process does not match the  
 494 file owner and acquire the appropriate privilege to change the file permission  
 495 bits of the file using *fchmod()*.

496                   **TEST:**    A call *fchmod(fildes, mode)* sets the *S\_ISUID* bit from the corresponding bit in  
497    the *mode* argument.  
498                   **TR:** For regular files: test all file permission bits.

499    For shared memory objects: test only the *S\_IRUSR*, *S\_IWUSR*, *S\_IRGRP*, *S\_IWGRP*,  
500    *S\_IROTH*, and *S\_IWOTH* file permission bits.  
501                   **ELSE NO\_TEST\_SUPPORT**  
502                   **ELSE NO\_OPTION**  
503                   *Conformance for fchmod:PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

504    **9**           **IF** *PCTS\_fchmod* and *PCTS\_CHMOD\_SGID* **THEN**  
505                   **SETUP:**    Open a file where the effective user ID of the calling process matches the file owner.  
506                   **TEST:**    A call *fchmod(fildes, mode)* sets the *S\_ISGID* bit from the corresponding bit in the  
507    *mode* argument.  
508                   **TR:** For regular files: test all file permission bits.

509    For shared memory objects: test only the *S\_IRUSR*, *S\_IWUSR*, *S\_IRGRP*, *S\_IWGRP*,  
510    *S\_IROTH*, and *S\_IWOTH* file permission bits.  
511                   **ELSE NO\_OPTION**  
512                   *Conformance for fchmod:PASS, NO\_OPTION*

513    **10**          **IF** *PCTS\_fchmod* and *PCTS\_CHMOD\_SGID* **THEN**  
514                    **IF** *PCTS\_GAP\_SGID\_fchmod* **THEN**  
515                      **SETUP:**    Open a file where the effective user ID of the calling process does not match the  
516    file owner and acquire the appropriate privilege to change the file permission  
517    bits of the file using *fchmod()*.  
518                      **TEST:**    A call *fchmod(fildes, mode)* sets the *S\_ISGID* bit from the corresponding bit in  
519    the *mode* argument.  
520                      **TR:** For regular files: test all file permission bits.

521    For shared memory objects: test only the *S\_IRUSR*, *S\_IWUSR*, *S\_IRGRP*, *S\_IWGRP*,  
522    *S\_IROTH*, and *S\_IWOTH* file permission bits.  
523                      **ELSE NO\_TEST\_SUPPORT**  
524                      **ELSE NO\_OPTION**  
525                      *Conformance for fchmod:PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

526    **11**          **IF** *PCTS\_fchmod* **THEN**  
527                    **IF** *PCTS\_RAP\_SGID\_fchmod* **THEN**  
528                      **SETUP:**    Open a regular such that the group ID of the file does not match the effective  
529    group ID or one of the supplementary group IDs of the process and one or more  
530    of the *S\_IXUSR*, *S\_IXGRP*, or *S\_IXOTH* bits of the file mode are set. Release  
531    appropriate privileges to change the *S\_ISGID* bit in the mode of the file.  
532                      **TEST:**    Bit *S\_ISGID* (set group ID on execution) in the mode of the file is cleared upon  
533    successful return from *fchmod()*.  
534                      **TR:** Test for each of the following bits being set individually in the file mode: *S\_IXUSR*,  
535    *S\_IXGRP*, and *S\_IXOTH*

536    **ELSE NO\_TEST\_SUPPORT**  
537                      **ELSE NO\_OPTION**  
538                      *Conformance for fchmod:PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

539     **12**     **IF** *PCTS\_RAP\_SGID\_chmod* **THEN**  
540             **SETUP:**   Open a regular such that the group ID of the file does not match the effective group  
541                           ID or one of the supplementary group IDs of the process and one or more of the  
542                           *S\_IXUSR*, *S\_IXGRP*, or *S\_IXOTH* bits of the file mode are set. Release appropriate  
543                           privileges to change the *S\_ISGID* bit in the mode of the file.  
544             **TEST:**     Bit *S\_ISGID* (set group ID on execution) in the mode of the file is cleared upon  
545                           successful return from *chmod()*.  
546             **TR:**     Test for each of the following bits being set individually in the file mode: *S\_IXUSR*,  
547                           *S\_IXGRP*, and *S\_IXOTH*  
548             **ELSE NO\_TEST\_SUPPORT**  
549             Conformance for *chmod*: *PASS, NO\_TEST\_SUPPORT*,

550     **13**     **IF** *PCTS\_fchmod* **THEN**  
551             **TEST:**     Upon successful completion, the *fchmod()* function marks for update the *st\_ctime*  
552                           field of the file.  
553             **ELSE NO\_OPTION**  
554             Conformance for *fchmod*: *PASS, NO\_OPTION*

555     **5.6.4.3 Returns**

556     **R\_1** **IF** *PCTS\_fchmod* **THEN**  
557             **TEST:**     Upon successful completion, the *fchmod()* function returns a value of zero.  
558             **ELSE NO\_OPTION**  
559             **SEE:**     Assertions in §5.6.4.2

560     **R\_2** **IF** *PCTS\_fchmod* **THEN**  
561             **TEST:**     Upon an unsuccessful completion, the *fchmod()* function returns a value of -1, sets  
562                           *errno* to indicate the error, and no change to the file *mode* occurs.  
563             **ELSE NO\_OPTION**  
564             **SEE:**     Assertions in §5.6.4.4

565     **5.6.4.4 Errors**

566     **14**     **IF** *PCTS\_fchmod* **THEN**  
567             **SETUP:**   Call *fchmod()* with the *fildev* argument not being a valid file descriptor.  
568             **TEST:**     A call to the *fchmod()* function on such a *fildev* returns a value of -1, sets *errno* to  
569                           [EBADF], and no change to the file *mode* occurs.  
570             **ELSE NO\_OPTION**  
571             Conformance for *fchmod*: *PASS, NO\_OPTION*

572     **15**     **IF** not *PCTS\_fchmod* **THEN**  
573             **TEST:**     A call to the *fchmod()* function on such a *fildev* returns a value of -1, sets *errno* to  
574                           [ENOSYS], and no change to the file *mode* occurs.  
575             **ELSE NO\_OPTION**  
576             Conformance for *fchmod*: *PASS, NO\_OPTION*

577     **16**     **IF** *PCTS\_fchmod* **THEN**  
578             **SETUP:**   The effective user ID does not match the owner of the file and the calling process  
579                           does not have the appropriate privileges.  
580             **TEST:**     A call to the *fchmod()* function on such a *fildev* returns a value of -1, sets *errno* to  
581                           [EPERM], and no change to the file *mode* occurs.  
582             **ELSE NO\_OPTION**  
583             Conformance for *fchmod*: *PASS, NO\_OPTION*

584     **17**     **IF** *PCTS\_fchmod* **THEN**  
585             **IF** *PCTS\_ROFS* **THEN**  
586                           **SETUP:**   Open a file that resides on a read-only file system.

587                   **TEST:**     A call to the *fchmod()* function on such a *fildev* returns a value of -1, sets *errno*  
 588                                   to [EROFS], and no change to the file *mode* occurs.

589                   **ELSE NO\_TEST\_SUPPORT**

590                   **ELSE NO\_OPTION**

591                   *Conformance for fchmod: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

592   **18**       **IF** *PCTS\_fchmod* and *PCTS\_EINVAL\_fchmod* **THEN**

593                   **SETUP:**    Open a file such that the *fildev* argument refers to a pipe.

594                   **TEST:**     A call to the *fchmod()* function on such a *fildev* returns a value of -1, sets *errno* to  
 595                                   [EINVAL], and no change to the file *mode* occurs.

596                   **ELSE NO\_OPTION**

597                   *Conformance for fchmod: PASS, NO\_OPTION*

## 598   **5.6.5 Change Owner and Group of a File**

599   Function: *chown()*

### 600   **5.6.5.1 Synopsis**

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

### 602   **5.6.5.2 Description**

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

### 604   **5.6.5.3 Returns**

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

### 606   **5.6.5.4 Errors**

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

## 608   **5.6.6 Set File Access and Modification Times**

609   Function: *utime()*

### 610   **5.6.6.1 Synopsis**

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

### 612   **5.6.6.2 Description**

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

### 614   **5.6.6.3 Returns**

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

### 616   **5.6.6.4 Errors**

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

## 618   **5.6.7 Truncate a File to a Specified Length**

619   Function: *ftruncate()*

620 **1**

621 *M\_GA\_stdC\_proto\_decl(int; ftruncate; int fildes, off\_t length; unistd.h;;)*

622 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3

623 *Conformance for ftruncate: PASS[1, 2], NO\_OPTION*

624 **2**

625 *M\_GA\_commonC\_int\_result\_decl(ftruncate; unistd.h;;)*

626 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3

627 *Conformance for ftruncate: PASS[1, 2], NO\_OPTION*

628 **3**

629 *M\_GA\_macro\_result\_decl(int; ftruncate; unistd.h;;)*

630 **SEE:** Assertion GA\_macro\_\_result\_decl in §1.3.4

631 *Conformance for ftruncate: PASS, NO\_OPTION*

632 **4**

633 *M\_GA\_macro\_args(ftruncate; unistd.h;;)*

634 **SEE:** Assertion GA\_macro\_\_args in §2.7.3

635 *Conformance for ftruncate: PASS, NO\_OPTION*

636 **5.6.7.2 Description**

637 **5** **IF** *PCTS\_ftruncate* **THEN**

638 **SETUP:** Open a regular file where the size of each file exceeds the *length* specified in

639 *ftruncate()* call to be tested.

640 **TEST:** The call *ftruncate(fildes, length)* causes the regular file known by the file descriptor

641 *fildes*, to be truncated to *length*.

642 **TR:** Test for write-only and read-write access to the file.

643 **ELSE** *NO\_OPTION*

644 *Conformance for fruncate: PASS, NO\_OPTION*

645 **D\_1** **IF** *PCTS\_ftruncate* and a PCD.1b documents the following **THEN**

646 **TEST:** A PCD.1b that documents whether the file is changed or its size increased after a call

647 *ftruncate(fildes, length)* if the file previously was smaller than *length* does so in

648 §5.6.7.2.

649 **ELSE** *NO\_OPTION*

650 *Conformance for fruncate: PASS, NO\_OPTION*

651 **6** **IF** *PCTS\_ftruncate* **THEN**

652 **IF** *PCTS\_EXTEND\_ON\_ftruncate* **THEN**

653 **SETUP:** Open a regular file whose size is less than the size to which it will be truncated.

654 **TEST:** The call *ftruncate(fildes, length)* where *length* is greater than the size of the file

655 causes the extended area to appear as if it were zero-filled.

656 **TR:** Test for write-only and read-write access to the file.

657 **ELSE** *NO\_TEST\_SUPPORT*

658 **ELSE** *NO\_OPTION*

659 *Conformance for fruncate: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

660 **7** **IF** *PCTS\_ftruncate* and *PCTS\_shm\_open* **THEN**

661 **TEST:** A call *ftruncate(fildes, length)* where *fildes* references a shared memory object, sets

662 the size of the shared memory object to *length*.

663 **TR:** Test for shared memory objects both larger and smaller than *length*.

664 read-write access

665 **ELSE** *NO\_OPTION*

666 *Conformance for fruncate: PASS, NO\_OPTION*

667 **D\_2** **IF** *PCTS\_ftruncate* and a PCD.1b documents the following **THEN**

668                   **TEST:**     A PCD.1b that documents the result of calling *ftruncate()* on a file that is not a regular  
669                                   file or a shared memory object does so in §5.6.7.2.  
670                   **ELSE NO\_OPTION**  
671                   *Conformance for ftruncate: PASS, NO\_OPTION*

672     **8**           **IF** *PCTS\_ftruncate* and *PCTS\_mmap* **THEN**  
673                   **IF** *\_POSIX\_MEMORY\_PROTECTION* **THEN**  
674                         **SETUP:**     Memory map a file so that the effect of *ftruncate()* is to decrease the size of a  
675                                   file and whole pages beyond the new end were previously mapped.  
676                         The whole mapped pages beyond the new end after a call to *ftruncate()* are discarded and  
677                                   references to them result in delivery of a SIGBUS signal.

678                                   **ELSE NO\_TEST\_SUPPORT**  
679                   **ELSE NO\_OPTION**  
680                   *Conformance for ftruncate: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

681     **9**           **IF** *PCTS\_ftruncate* and *PCTS\_shm\_open* and *PCTS\_mmap* **THEN**  
682                   **IF** *\_POSIX\_MEMORY\_PROTECTION* **THEN**  
683                         **SETUP:**     Memory map a shared memory object so that the effect of *ftruncate()* is to  
684                                   decrease the size of the shared memory object and whole pages beyond the new  
685                                   end were previously mapped.  
686                         The whole mapped pages beyond the new end after a call to *ftruncate()* are discarded and  
687                                   references to them result in delivery of a SIGBUS signal.  
688                                   **ELSE NO\_TEST\_SUPPORT**  
689                   **ELSE NO\_OPTION**  
690                   *Conformance for ftruncate: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

691     **10**          **IF** *PCTS\_ftruncate* **THEN**  
692                         **TEST:**     The value of the seek pointer is not to be modified by a ll to *ftruncate()*.  
693                   **ELSE NO\_OPTION**  
694                   *Conformance for ftruncate: PASS, NO\_OPTION*

695     **11**          **IF** *PCTS\_ftruncate* **THEN**  
696                         **TEST:**     Upon successful completion, the *ftruncate()* function marks for update the *st\_ctime*  
697                                   and *st\_mtime* fields of the file.  
698                   **ELSE NO\_OPTION**  
699                   *Conformance for ftruncate: PASS, NO\_OPTION*

700     **12**          **IF** *PCTS\_ftruncate* **THEN**  
701                         **TEST:**     The file is unaffected by an unsuccessful call to *ftruncate()*.  
702                         **TR:**     Test the size of the file as well as the *st\_ctime* and *st\_mtime* fields of the file.  
703                   **ELSE NO\_OPTION**  
704                   *Conformance for ftruncate: PASS, NO\_OPTION*

### 705     **5.6.7.3 Returns**

706     **R\_1** **IF** *PCTS\_ftruncate* **THEN**  
707                         **TEST:**     Upon successful completion, the *ftruncate()* function returns to zero.  
708                   **ELSE NO\_OPTION**  
709                   **SEE:**     Assertions in §5.6.7.2.

710     **R\_2** **IF** *PCTS\_ftruncate* **THEN**  
711                         **TEST:**     An unsuccessful call to the *ftruncate()* function returns -1 and sets *errno* to indicate  
712                                   the error.  
713                   **ELSE NO\_OPTION**  
714                   **SEE:**     Assertions in §5.6.7.4.

### 715     **5.6.7.4 Errors**

- 716 **13** **IF** *PCTS\_ftruncate* **THEN**  
 717 **TEST:** A call to *ftruncate()* where the *filides* argument is not a valid file descriptor open for  
 718 writing returns -1 and sets *errno* to [EBADF].  
 719 **TR:** Test the size of the file as well as the *st\_ctime* and *st\_mtime* fields of the file.  
 720 **ELSE NO\_OPTION**  
 721 *Conformance for ftruncate: PASS, NO\_OPTION*
- 722 **14** **IF** *PCTS\_ftruncate* **THEN**  
 723 **TEST:** A call to *ftruncate()* where the *filides* argument does not refer to a file on which this  
 724 operation is possible returns -1 and sets *errno* to [EINVAL]  
 725 **ELSE NO\_OPTION**  
 726 *Conformance for ftruncate: PASS, NO\_OPTION*
- 727 **15** **IF** *PCTS\_ftruncate* **THEN**  
 728 **TEST:** A call to *ftruncate()* where the file resides on a read-only file system returns -1 and  
 729 sets *errno* to [EROFS]  
 730 **ELSE NO\_OPTION**  
 731 *Conformance for ftruncate: PASS, NO\_OPTION*

## 732 5.7 Configurable Pathname Variables

### 733 5.7.1 Get Configurable Pathname Variables

734 Functions: *pathconf()*, *fpathconf()*

#### 735 5.7.1.1 Synopsis

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

#### 737 5.7.1.2 Description

- 738 **1** **SETUP:** Include the header `<unistd.h>`  
 739 **TEST:** The constants `{_PC_ASYNC_IO}`, and `{_PC_SYNC_IO}` are defined.  
 740 *Conformance for pathconf: PASS*
- 741 **2** **FOR:** *pathconf()*, and *fpathconf()*  
 742 **IF** `{_POSIX_ASYNC_IO}` is defined when `<unistd.h>` is included **THEN**  
 743 **SETUP:** Include the header `<unistd.h>`  
 744 **TEST:** A call *function()* with a *name* parameter equal to `{_POSIX_ASYNC_IO}` returns  
 745 a value equal to `{_POSIX_ASYNC_IO}`.  
 746 **ELSE NO\_OPTION**  
 747 *Conformance for pathconf, fpathconf: PASS, NO\_OPTION*
- 748 **3** **FOR:** *pathconf()*, and *fpathconf()*  
 749 **SETUP:** Include the header `<unistd.h>`  
 750 **TEST:** A call *function()* with a *name* parameter equal to `{_POSIX_ASYNC_IO}` that refers to a file  
 751 other than a directory returns a value corresponding to the option `{_POSIX_ASYNC_IO}` for  
 752 that file.  
 753 *Conformance for pathconf, fpathconf: PASS*
- 754 **4** **FOR:** *pathconf()*, and *fpathconf()*  
 755 **IF** `{_POSIX_PRIO_IO}` is defined when `<unistd.h>` is included **THEN**  
 756 **SETUP:** Include the header `<unistd.h>`  
 757 **TEST:** A call *function()* with a *name* parameter equal to `{_POSIX_PRIO_IO}` returns a  
 758 value equal to `{_POSIX_PRIO_IO}`.  
 759 **ELSE NO\_OPTION**  
 760 *Conformance for pathconf, fpathconf: PASS, NO\_OPTION*



- 761 **5**       **FOR:**    *pathconf()*, and *fpathconf()*  
762       **SETUP:** Include the header `<unistd.h>`  
763       **TEST:**    A call *function()* with a *name* parameter equal to `{_POSIX_PRIO_IO}` that refers to a file  
764       other than a directory returns a value corresponding to the option `{_POSIX_PRIO_IO}` for that  
765       file.  
766       *Conformance for pathconf, fpathconf: PASS*
- 767 **6**       **FOR:**    *pathconf()*, and *fpathconf()*  
768       **IF** `{_POSIX_SYNCH_IO}` is defined when `<unistd.h>` included **THEN**  
769       **SETUP:** Include the header `<unistd.h>`  
770       **TEST:**    A call *function()* with a *name* parameter equal to `{_PC_SYNC_IO}` returns a value equal to  
771       `{_PC_SYNC_IO}`  
772       **ELSE** *NO\_OPTION*  
773       *Conformance for pathconf, fpathconf: PASS, NO\_OPTION*
- 774 **7**       **FOR:**    *pathconf()*, and *fpathconf()*  
775       **SETUP:** Include the header `<unistd.h>`  
776       **TEST:**    A call *function()* with a *name* parameter equal to `{_PC_SYNC_IO}` that refers to a file other  
777       than a directory returns a value corresponding to the option `{_POSIX_SYNC_IO}` for that file.  
778       *Conformance for pathconf, fpathconf: PASS*
- 779 **D\_1** **IF** a PCD.lb documents the following **THEN**  
780       **TEST:**    A PCD.lb that documents whether an implementation supports an association of the  
781       variable name with the specified file if the *path* argument to *pathconf()* or *fildev*  
782       argument to *pathconf()* refers to a directory does so in §5.7.1.3.  
783       **ELSE** *NO\_OPTION*  
784       *Conformance for pathconf, fpathconf: PASS, NO\_OPTION*
- 785 **5.7.1.3 Returns**
- 786 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.
- 787 **5.7.1.4 Errors**
- 788 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b {3} assertions.

This page is intentionally blank.

## Section 6: Input and Output Primitives

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

### 181 **6.1 Pipes**

#### 182 **6.1.1 Create an Inter-Process Channel**

183 Function: *pipe()*

##### 184 **6.1.1.1 Synopsis**

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

##### 186 **6.1.1.2 Description**

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

##### 188 **6.1.1.3 Returns**

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

##### 190 **6.1.1.4 Errors**

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

### 192 **6.2 File Descriptor Manipulation**

#### 193 **6.2.1 Duplicate an Open File Descriptor**

194 Function: *dup()*, *dup2()*

##### 195 **6.2.1.1 Synopsis**

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

##### 197 **6.2.1.2 Description**

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

##### 199 **6.2.1.3 Returns**

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

##### 201 **6.2.1.4 Errors**

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

## 203 **6.3 File Descriptor Deassignment**

### 204 **6.3.1 Close a File**

205 Function: *close()*

#### 206 **6.3.1.1 Synopsis**

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

#### 208 **6.3.1.2 Description**

209 **1** **IF** {\_POSIX\_ASYNCHRONOUS\_IO} **THEN**  
 210 **TEST:** An asynchronous I/O operation that is not canceled completes as if the *close()*  
 211 operation had not yet occurred.  
 212 **ELSE NO\_OPTION**  
 213 *Conformance for close: PASS, NO\_TEST, NO\_OPTION*

214 **2** **IF** {\_POSIX\_ASYNCHRONOUS\_IO} **THEN**  
 215 **TEST:** All operations that are not canceled complete as if the *close()* blocked until the  
 216 operations completed  
 217 **ELSE NO\_OPTION**  
 218 *Conformance for close: PASS, NO\_TEST, NO\_OPTION*

219 **D-1** **TEST:** The PCD.1b documents whether any I/O operations is canceled, and which I/O operation may  
 220 be canceled upon a call to the *close()* function in §6.3.1.2  
 221 *Conformance for close: PASS*

222 **3** **IF** *PCTS\_shm\_open* and *PCTS\_mmap* **THEN**  
 223 **SETUP:** Create and map a memory object that remains referenced at the last close.  
 224 **TEST:** After a call to *close()* the entire contents of the memory object persist until the  
 225 memory object becomes unreferenced.  
 226 **TR:** Test for shared memory.  
 227  
 228 If {POSIX\_MAPPED\_FILES}; test for memory mapped files.  
 229 **ELSE NO\_OPTION**  
 230 *Conformance for close: PASS, NO\_OPTION*

231 **4** **IF** *PCTS\_shm\_open* and *PCTS\_mmap* **THEN**  
 232 **SETUP:** Create a memory object and map it in at least two processes. Then unlink the memory  
 233 object. Perform the last close of the memory object and such that the close results in  
 234 the memory object becoming unreferenced.  
 235 **TEST:** A call to *close()* removes the memory object.  
 236 **ELSE NO\_OPTION**  
 237 *Conformance for close: PASS, NO\_OPTION*

#### 238 **6.3.1.3 Returns**

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

#### 240 **6.3.1.4 Errors**

241  
 242 There are only IEEE Std 2003.1-1992 {4} assertions in this subclause; no POSIX.1b{3} assertions.

243

## 244 **6.4 Input and Output**

### 245 **6.4.1 Read from a File**

246 Function: *read()*

247 **6.4.1.1 Synopsis**

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

249 **6.4.1.2 Description**

- 250 **1 IF** *PCTS\_read* and {POSIX\_SYNCHRONIZED\_IO} **THEN**
- 251 **SETUP:** Open a file by calling *open()* with O\_RSYNC and O\_DSYNC set in the *oflag* parameter.
- 252 **TEST:** A read operation initiated by calling *read()* either completes by transferring an image
- 253 of the data to the requesting process or, if unsuccessful, by diagnosing and returning
- 254 an indicator of the error.
- 255 **TR:** Test for regular files and, if PCTS\_GTI\_DEVICE, terminals.
- 256 **NOTE:** There is no known portable test method for this assertion.
- 257 **ELSE NO\_OPTION**
- 258 **SEE:** Assertion GA\_syncIODataIntegrityRead in §2.2.2.119
- 259 *Conformance for read: PASS, NO\_TEST, NO\_OPTION*
- 260 **2 IF** *PCTS\_read* and {POSIX\_SYNCHRONIZED\_IO} **THEN**
- 261 **SETUP:** Open a file by calling *open()* with O\_RSYNC and O\_DSYNC set in the *oflag* parameter
- 262 **TEST:** At the time that the synchronized read operation initiated by calling *read()* occurs, any
- 263 pending write requests affecting the data to be read are written to the physical medium
- 264 containing the file prior to reading the data.
- 265 **TR:** Test for regular files.
- 266 **NOTE:** There is no known portable test method for this assertion.
- 267 **ELSE NO\_OPTION**
- 268 **SEE:** Assertion GA\_syncIODataIntegrityWbeforeR in §2.2.2.119
- 269 *Conformance for read: PASS, NO\_TEST, NO\_OPTION*
- 270 **3 IF** *PCTS\_read* and {POSIX\_SYNCHRONIZED\_IO} **THEN**
- 271 **SETUP:** Open a file by calling *open()* with O\_RSYNC and O\_DSYNC set in the *oflag* parameter
- 272 **TEST:** At the time that the synchronized read operation initiated by calling *read()* occurs, any
- 273 pending write request affecting the data to be read are written to the physical medium
- 274 containing the file prior to reading the data and the following file attributes are also
- 275 written to the physical medium containing the file prior to returning to the calling
- 276 process:
- 277 1. File mode.
- 278 2. File serial number.
- 279 3. ID of device containing this file.
- 280 4. Number of links.
- 281 5. User ID of the owner of the file.
- 282 6. Group ID of the group of the file.
- 283 7. The file size in bytes.
- 284 8. Time of last access.
- 285 9. Time of last data modification.
- 286 10. Time of last file status change.
- 287 **TR:** Test for regular files.
- 288 **NOTE:** There is no known portable test method for this assertion.
- 289 **ELSE NO\_OPTION**
- 290 **SEE:** Assertion GA\_syncIOFileIntegrityRead in §2.2.2.120

291 *Conformance for read: PASS, NO\_TEST, NO\_OPTION*

292 **D-1** **IF** a PCD.1b documents the following **THEN**

293 **TEST:** A PCD.1b that documents the result of a call to the *read()* function when *fildev* refers

294 to a shared memory object does so in §6.4.1.2.

295 **ELSE NO\_OPTION**

296 *Conformance for read: PASS, NO\_OPTION*

297 **6.4.1.3 Returns**

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

299 **6.4.1.4. Errors**

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

301 **6.4.2 Write to a File**

302 Function: *write()*

303 **6.4.2.1. Synopsis**

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

305 **6.4.2.2. Description**

306 **1** **IF** *PCTS\_write* and {POSIX\_SYNCHRONIZED\_IO} **THEN**

307 **SETUP:** Open a file by calling *open()* with O\_DSYNC set in the *oflag* parameter.

308 **TEST:** A write operation initiated by calling *write()* either completes by transferring an image

309 of the data to the physical medium containing the file or, if unsuccessful, by

310 diagnosing and returning an indicator of the error.

311 **TR:** Test for regular files and, if PCTS\_GTI\_DEVICE, terminals.

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

313 **ELSE NO\_OPTION**

314 **SEE:** Assertion GA\_syncIODataIntegrityWrite in §2.2.2.119

315 *Conformance for write: PASS, NO\_TEST, NO\_OPTION*

316 **2** **IF** *PCTS\_write* and {POSIX\_SYNCHRONIZED\_IO} *PCTS\_write* and {POSIX\_SYNCHRONIZED\_IO} **THEN**

317 **SETUP:** Open a file by calling *open()* with O\_SYNC set in the *oflag* parameter.

318 **TEST:** At the time that the synchronized write operation initiated by calling *write()* occurs,

319 the data are written to the physical medium containing the file and the following file

320 attributes are also written to the physical medium containing the file prior to returning

321 to the calling process:

322 1. File mode.

323 2. File serial number.

324 3. ID of the device containing this file.

325 4. Number of links.

326 5. User ID of the owner of the file.

327 6. Group ID of the group of the file.

328 7. The file size in bytes.

329 8. Time of last access.

330 9. Time of last data modification.

331 10. Time of last file status change.

332 **TR:** Test for regular files.

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

334 **ELSE NO\_OPTION**

335 **SEE:** Assertion GA\_syncIOFileIntegrityWrite in §2.2.2.120

336 *Conformance for write: PASS, NO\_TEST, NO\_OPTION*

337 **D-1 IF** a PCD.1b documents the following **THEN**

338 **TEST:** A PCD.1b that documents the result of a call to the *write()* function when *files* refers  
339 to a shared memory object does so in §6.4.2.2

340 **ELSE NO\_OPTION**

341 *Conformance for write: PASS, NO\_OPTION*

### 342 6.4.2.3 Returns

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

### 344 6.4.2.4 Errors

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

## 346 6.5 Control Operations on Files

### 347 6.5.1 Data Definitions for File Control Operations

348 **1 SETUP:** Include the header `<fcntl.h>`

349 **TEST:** The constants `O_DSYNC`, `O_RSYNC`, and `O_SYNC` are defined and are unique values with  
350 respect to each other and `O_CREAT`, `O_EXCL`, `O_NOCTTY`, `O_TRUNC`, `O_APPEND`,  
351 `O_NONBLOCK`, `O_RDONLY`, `O_RDWR`, AND `O_WRONLY`.

352 *Conformance for fcntl.h: PASS*

353 NOTE: IEEE Std 2003.1-1992 {4} was not as explicit as the above assertion regarding the uniqueness of the  
354 constants,

### 355 6.5.2 File Control

356 Function: *fcntl()*

#### 357 6.5.2.1 Synopsis

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

#### 359 6.5.2.2. Description

360 All of the assertions for the Description subclause of *fcntl()* except for those relating to file locking and setting file  
361 status flags were rewritten from IEEE Std 2003.1-1992 {4} because of the requirement that they apply to shared  
362 memory objects and the change in the form of assertions between IEEE Std 2003.1-1992 {4} and this standard.

363 **(IEEE Std 2003.1-1992 {4})04**

364 *UNUSED*

365 **(IEEE Std 2003.1-1992 {4})05**

366 *UNUSED*

367 **(IEEE Std 2003.1-1992 {4})06**

368 *UNUSED*

- 369 **IEEE Std 2003.1-1992 {4}07**  
370 *UNUSED*
- 371 **IEEE Std 2003.1-1992 {4}08**  
372 *UNUSED*
- 373 **IEEE Std 2003.1-1992 {4}09**  
374 *UNUSED*
- 375 **IEEE Std 2003.1-1992 {4}10**  
376 *UNUSED*
- 377 **IEEE Std 2003.1-1992 {4}11**  
378 *UNUSED*
- 379 **IEEE Std 2003.1-1992 {4}12**  
380 *UNUSED*
- 381 **4**       **TEST:**    A call *fcntl(fildes, F\_DUPFD, arg)* creates a new file descriptor that:
- 382                   1.    Is the lowest numbered one available greater than or equal to the argument *arg*.
- 383                   2.    Refers to the same open file description (file pointer, access mode, and file status  
384 flags) as the original file descriptor.
- 385                   3.    Shares the same locks as the original file descriptor, if *fildes* does not refer to a  
386 shared memory object.
- 387                   4.    Has the FD\_CLOEXEC flag clear.
- 388 **TR:** Test for a regular file. If *PCTS\_shm\_open*: Test for *fildes* referring to a shared memory object.  
389 *Conformance for fcntl: PASS*
- 390 **5**       **TEST:**    A call *fcntl(fildes, F\_GETFD)* returns the status of the file descriptor flags.  
391 **TR:** Test for a regular file. If *PCTS\_shm\_open*: Test for *fildes* referring to a shared memory object.  
392 *Conformance for fcntl: PASS*
- 393 **6**       **FOR:**    *execl(), execv(), execl(), execve(), execlp()* and *execvp()*  
394 **TEST:**    A call *fcntl(fildes, F\_SETFD, arg)* where the FD\_CLOEXEC flag in *arg* is nonzero sets the  
395 close-on-exec flag for the file associated with *fildes* and *function()*.  
396 **TR:** Test for a regular file. If *PCTS\_shm\_open*: Test for *fildes* referring to a shared memory object.  
397 *Conformance for fcntl: PASS*
- 398 **7**       **FOR:**    *execl(), execv(), execl(), execve(), execlp(),* and *execvp()*  
399 **TEST:**    A call *fcntl(fildes, F\_SETFD, arg)* where the FD\_CLOEXEC flag in *arg* is nonzero sets the  
400 close-on-exec flag for the file associated with *fildes* and *function()*.  
401 **TR:** Test for a regular file. If *PCTS\_shm\_open*: Test for *fildes* referring to a shared memory object.  
402 *Conformance for fcntl: PASS*
- 403 **8**       **TEST:**    A call *fcntl(fildes, F\_GETFL)* returns the file status flags O\_APPEND, O\_DSYNC, O\_NONBLOCK,  
404 O\_RSYNC, and O\_SYNC and the file access modes O\_RDONLY, O\_RDWR and O\_WRONLY.  
405 **TR:** Test for a regular file. If *PCTS\_shm\_open*: Test for *fildes* referring to a shared memory object.  
406 *Conformance for fcntl: PASS*
- 407 **9**       **TEST:**    A call *fcntl(fildes, F\_SETFL, 0)* sets the status flags for the file referenced by *fildes* to 0.  
408 **TR:** Test for regular files  
409 *Conformance for fcntl: PASS*
- 410 **10**      **TEST:**    A call *fcntl(fildes, F\_SETFL, arg)* sets the status flags for the file referenced by *fildes* to  
411 values in *arg*.



412           **TR:** Test with *arg* having the values O\_APPEND, and O\_NONBLOCK, individually and together, for  
413           regular files.  
414                 If {\_POSIX\_SYNCHRONIZED\_IO}: Test with *arg* having the values O\_APPEND, O\_DSYNC,  
415                 O\_NONBLOCK, O\_RSUNC and O\_SYNC, individually and all together, on a file supporting  
416                 {\_POSIX\_SYNCHRONIZED\_IO}.  
417           *Conformance for fcntl: PASS*

418   **D-1     IF** *PCTS\_shm\_open* and PCD.1b that documents the following **THEN**  
419           **TEST:**     A PCD.1b that documents the effect of the values F\_SETFL, F\_GETLK, F\_SETLK, and  
420                         F\_SETLKW. for the argument *cmd* of the function *fcntl()* when the file descriptor *fd*  
421                         refers to a shared member object, does so in §6.5.2.2  
422           **ELSE NO\_OPTION**  
423           *Conformance for fcntl: PASS, NO\_OPTION.*

424   **6.5.2.3   Returns**

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

426   **6.5.2.4   Errors**

427   **(IEEE Std 2003.1-1992 {4})41**  
428           *UNUSED*

429   **(IEEE Std 2003.1-1992 {4})42**  
430           *UNUSED*

431   **(IEEE Std 2003.1-1992 {4})48**  
432           *UNUSED*

433   **(IEEE Std 2003.1-1992 {4})49**  
434           *UNUSED*

435   **41        TEST:**     A call *fcntl(fd, F\_DUPFD, arg)* where *arg* is negative returns -1 and sets *errno* to  
436                         [EINVAL].  
437           **TR:** Test for a regular file. If *PCTS\_shm\_open*: Test for *fd* referring to a shared memory object.  
438           *Conformance for fcntl: PASS*

439   **42        IF** a call *sysconf()* with an argument of {\_SC\_OPEN\_MAX} does not return -1  
440           **THEN**  
441                 **TEST:**     A call *fcntl(fd, F\_DUPFD, arg)* where *arg* is greater than or equal to [OPEN\_MAX]  
442                         returns -1 and sets *errno* to [EINVAL].  
443                 **TR:** Test for a regular file. If *PCTS\_shm\_open*: Test for *fd* referring to a shared memory  
444                         object.  
445                 **ELSE NO\_TEST\_SUPPORT**  
446                 *Conformance for fcntl: PASS, NO\_TEST\_SUPPORT*

447   **42.1 IF** a call *sysconf()* with an argument of {\_SC\_OPEN\_MAX} returns -1 **THEN**  
448                 **TEST:**     A call *fcntl(fd, F\_DUPFD, arg)* where *arg* is equal to [PCTS\_OPEN\_MAX]  
449                         returns a valid file descriptor and does not set *errno* to [EINVAL].  
450                 **TR:** Test for a regular file. If *PCTS\_shm\_open*: Test for *fd* referring to a shared memory  
451                         object.  
452                 **ELSE NO\_TEST\_SUPPORT**  
453                 *Conformance for fcntl: PASS, NO\_TEST\_SUPPORT*

454   **42.2 IF** *PCTS\_NO\_SYNC\_IO\_FILE* **THEN**  
455                 **TEST:**     A call *fcntl(fd, F\_SETL, arg)* where any of the file status flags O\_DSYNC, O\_RSYNC,  
456                         OR O\_SYNC are set in the *arg* argument returns -1 and sets *errno* to [EINVAL].  
457                 **TR:** Test for a file that does not support synchronized I/O.  
458                 **ELSE NO\_OPTION**

459 *Conformance for fcntl: PASS, NO\_OPTION*

460 **48** **IF** {OPEN\_MAX} ≤ PCTS\_OPEN\_MAX **THEN**

461 **SETUP:** Open {OPEN\_MAX} files.

462 **TEST:** A call *fcntl(fildes, F\_DUPD, arg)* returns -1 and sets *errno* to [EMFILE].

463 **TR:** Test for a regular file. If PCTS\_shm\_open: Test for *fildes* referring to a shared

464 memory object.

465 **ELSE NO\_TEST\_SUPPORT**

466 *Conformance for fcntl: PASS, NO\_TEST\_SUPPORT*

467 **48.1** **IF** {OPEN\_MAX} > PCTS\_OPEN\_MAX **THEN**

468 **SETUP:** Open PCTS\_OPEN\_MAX files.

469 **TEST:** A call *fcntl(fildes, F\_DUPD, arg)* returns a valid file descriptor and does not set *errno*

470 to [EMFILE].

471 **TR:** Test for a regular file. If PCTS\_shm\_open: Test for *fildes* referring to a shared memory

472 object.

473 **ELSE NO\_TEST\_SUPPORT**

474 *Conformance for fcntl: PASS, NO\_TEST\_SUPPORT*

475

476 **49** **IF** {OPEN\_MAX} ≤ PCTS\_OPEN\_MAX **THEN**

477 **SETUP:** Open {OPEN\_MAX} files then make a file descriptor available whose value will be

478 less than that specified in the *arg* support to *fcntl()*.

479 **TEST:** A call *fcntl(fildes, F\_DUPD, arg)* where *arg* is greater than the lowest available file

480 descriptor returns -1 and sets *errno* to [EMFILE].

481 **TR:** Test for a regular file. If PCTS\_shm\_open: Test for *fildes* referring to a shared memory

482 object.

483 **ELSE NO\_TEST\_SUPPORT**

484 *Conformance for fcntl: PASS, NO\_TEST\_SUPPORT*

485 **49.1** **IF** {OPEN\_MAX} > PCTS\_OPEN\_MAX **THEN**

486 **SETUP:** Open PCTS\_OPEN\_MAX files then make a file descriptor available whose value will be

487 less than that specified in the *arg* argument to *fcntl()*.

488 **TEST:** A call *fcntl(fildes, F\_DUPD, arg)* returns a valid file descriptor greater than or equal

489 to the *arg* argument and does not set *errno* to [EMFILE].

490 **TR:** Test for a regular file. If PCTS\_shm\_open: Test for *fildes* referring to a shared memory

491 object.

492 **ELSE NO\_TEST\_SUPPORT**

493 *Conformance for fcntl: PASS, NO\_TEST\_SUPPORT*

494 **6.5.3 Reposition Read/Write File Offset**

495 Function: *lseek()*

496 **6.5.3.1 Synopsis**

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

498 **6.5.3.2 Description**

499 **D-1** **IF** a PCD.1b documents the following **THEN**

500 **TEST:** A PCD.1b that documents the result of calling the *lseek()* function when *fildes* refers

501 to a shared memory object does so in §6.5.3.2

502 **ELSE NO\_OPTION**

503 *Conformance for lseek: PASS, NO\_OPTION*

504 **6.5.3.3. Returns**

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

506 **6.5.3.4. Errors**

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

508 **6.6 File Synchronization**

509 **D-1 TEST:** The PCD.1b documents the hardware characteristics upon which the implementation relies  
 510 to assure that data is successfully transferred for synchronized I/O operation in §6.6  
 511 *Conformance for syncio: PASS*

512 **6.6.1 Synchronize the State of a File**513 Function: *fsync()*514 **6.6.1.1 Synopsis**

515 **1**  
 516 *M\_GA\_stdC\_proto\_decl(int;fsync; intfildes; unistd.h;;;)*  
 517 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 518 *Conformance for fsync: PASS[1, 2], NO\_OPTION*

519 **2**  
 520 *M\_GA\_commonC\_int\_result\_decl(fsync; unistd.h;;;)*  
 521 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 522 *Conformance for fsync: PASS[1, 2], NO\_OPTION*

523 **3**  
 524 *M\_GA\_macro\_result\_decl(int; fsync; unistd.h;;;)*  
 525 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 526 *Conformance for fsync: PASS, NO\_OPTION*

527 **4**  
 528 *M\_GA\_macro\_args (fsync; unistd.h;;;)*  
 529 **SEE** Assertion GA\_macro\_args in §2.7.3  
 530 *Conformance for fsync: PASS, NO\_OPTION*

531 **6.6.1.2 Description**

532 **D-1 IF** PCTS\_ *fsync* **THEN**  
 533 **TEST:** The PCD.1b documents the manner in which the *fsync()* function transfers all data to  
 534 the storage device associated with the open file description specified by the *fildes*  
 535 argument in §6.6.1.2  
 536 **ELSE** *NO\_OPTION*

537 **5 IF** PCTS\_ *fsync* **THEN**  
 538 **SETUP:** Write data to a file so that an error condition will not exist when *fsync()* is called.  
 539 **TEST:** The *fsync()* function does not return until the system has completed transferring  
 540 all data to the storage device associated with the open file description specified  
 541 by the *fildes* argument.  
 542 **NOTE:** There is no known portable test method for this assertion.  
 543 **ELSE** *NO\_OPTION*  
 544 *Conformance for fsync: PASS, NO\_TEST, NO\_OPTION*

545 **R-1 IF** PCTS\_ *fsync* **THEN**  
 546 **TEST:** A call to the *fsync()* function returns an error that is detected.  
 547 **NOTE:** There is no known portable test method for this assertion.  
 548 **ELSE** *NO\_OPTION*  
 549 **SEE:** Assertions in §6.6.1.4

550 **D-2** **IF** PCTS\_ *fsync* **THEN**

551 **TEST:** The PCD.1b documents sufficient information for the user to determine whether it is

552 possible to configure an application and installation to ensure that the data is stored

553 with the degree of required stability for the intended use of the *fsync()* function in

554 §6.6.1.2.

555 **ELSE NO\_OPTION**

556 *Conformance for fsync: PASS, NO\_OPTION*

557 **6** **IF** PCTS\_ *fsync* and {POSIX\_SYNCHRONIZED\_IO} **THEN**

558 **SETUP:** Open a file for read-write access without specifying the O\_DSYNC, O\_RSYNC, or

559 O\_DSYNC in the *oflags* argument of *open()*. Then write some data to the file and read

560 from another place in the file.

561 **TEST:** A call to the *fsync()* function forces all currently queued I/O operations associated

562 with the file indicated by file descriptor *filde*s to the synchronized I/O file integrity

563 completion state. That is any pending write requests affecting the data to be read are

564 written to the physical medium containing the file prior to reading the data and the

565 following file attributes are also written to the physical medium containing the file

566 prior to returning to the calling process:

567 1. File mode.

568 2. File serial number.

569 3. ID of device containing this file.

570 4. Number of links.

571 5. User ID of the owner of the file.

572 6. Group ID of the group of the file.

573 7. The file size in bytes.

574 8. Time of last access.

575 9. Time of last data modification.

576 10. Time of last file status change.

577 **NOTE:** There is no known portable test method for this assertion,

578 **ELSE NO\_OPTION**

579 *Conformance for fsync: PASS, NO\_TEST, NO\_OPTION*

580 **7** **IF** PCTS\_ *fsync* and {POSIX\_SYNCHRONIZED\_IO} **THEN**

581 **SETUP:** Open a file for read-write access without specifying the O\_DSYNC, O\_RSYNC, or

582 O\_DSYNC in the *oflags* argument of *open()*. Then write some data to the file..

583 **TEST:** A call to the *fsync()* function forces all currently queued I/O operations associated

584 with the file indicated by file descriptor *filde*s to the synchronized I/O file integrity

585 completion state. That is the data are written to the physical medium containing the

586 file and the following file attributes are also written to the physical medium

587 containing the file prior to returning to the calling process:

588 1. File mode.

589 2. File serial number.

590 3. ID of device containing this file.

591 4. Number of links.

592 5. User ID of the owner of the file.

- 593 6. Group ID of the group of the file.
- 594 7. The file size in bytes.
- 595 8. Time of last access.
- 596 9. Time of last data modification.
- 597 10. Time of last file status change.
- 598 **NOTE:** There is no known portable test method for this assertion,
- 599 **ELSE NO\_OPTION**
- 600 *Conformance for fsync: PASS, NO\_TEST, NO\_OPTION*
- 601 **6.6.1.3 Returns**
- 602 **R-2 IF PCTS\_fsync THEN**
- 603 **TEST:** Upon successful completion, the *fsync()* function returns zero.
- 604 **ELSE NO\_OPTION**
- 605 **SEE:** Assertions in §6.6.1.2
- 606 **8 TEST:** Upon unsuccessful completion, the *fsync()* returns -1 and set *errno* to indicate the error.
- 607 **SEE:** Assertions in §6.6.1.4
- 608 *Conformance for fsync: PASS*
- 609 **6.6.1.4 Errors**
- 610 **9 IF PCTS\_fsync THEN**
- 611 **SETUP:** Call the *fsync()* function with the *fildev* argument not being a valid file descriptor.
- 612 **TEST:** *fsync()* returns -1 and set *errno* to [EBADF].
- 613 **ELSE NO\_OPTION**
- 614 *Conformance for fsync: PASS, NO\_OPTION*
- 615 **10 IF PCTS\_fsync THEN**
- 616 **IF PCTS\_NO\_SYNC\_IO\_FILE THEN**
- 617 **SETUP:** Call the *fsync()* function with the *fildev* argument pointing to a file that does
- 618 not support synchronized I/O
- 619 **TEST:** *fsync()* returns -1 and set *errno* to [EINVAL].
- 620 **ELSE NO\_TEST\_SUPPORT**
- 621 **ELSE NO\_OPTION**
- 622 *Conformance for fsync: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 623 **11 IF not PCTS\_fsync THEN**
- 624 **SETUP:** Call the *fsync()* function even though it is not supported by this implementation
- 625 **TEST:** *fsync()* returns -1 and set *errno* to [ENOSYS].
- 626 **ELSE NO\_OPTION**
- 627 *Conformance for fsync: PASS, NO\_OPTION*
- 628 **12 IF PCTS\_fsync THEN**
- 629 **TEST:** In the event that any of the queued I/O operations fail, *fsync()* returns the error
- 630 conditions defined for *read()* and *write()*.
- 631 **ELSE NO\_OPTION**
- 632 *Conformance for fsync: PASS, NO\_TEST, NO\_OPTION*

## 633 6.6.2 Synchronize the Data of a File

634 Function: *fdatasync()*

### 635 6.6.2.1 Synopsis

636 **1**  
637 *M\_GA\_stdC\_Proto\_decl(int; fdasync: int fildes; unistd.h;;)*  
638 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
639 *Conformance for fdasync: PASS[1, 2] NO\_OPTION*

640 **2**  
641 *M\_GA\_commonC\_int\_result\_decl(int; fdasync: int fildes; unistd.h;;)*  
642 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
643 *Conformance for fdasync: PASS[1, 2] NO\_OPTION*

644 **3**  
645 *M\_GA\_macro\_result\_decl(int; fdasync: int fildes; unistd.h;;)*  
646 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
647 *Conformance for fdasync: PASS[1, 2] NO\_OPTION*

648 **4**  
649 *M\_GA\_macro\_args(int; fdasync: int fildes; unistd.h;;)*  
650 **SEE:** Assertion GA\_macro\_result\_decl in §2.7.3  
651 *Conformance for fdasync: PASS[1, 2] NO\_OPTION*

652 **6.6.2.2 Description**

653 **5** **IF** PCTS\_fdasync **THEN**  
654 **SETUP:** Write data to a file so that an error condition will not exist when *fdasync()* is called.  
655 **TEST:** The *fdasync()* function does not return until the system has completed  
656 transferring all data to the storage device associated with the open file  
657 description specified by the *fildes* argument.  
658 **NOTE:** There is no known portable test method for this assertion.  
659 **ELSE NO\_OPTION**  
660 *Conformance for fdasync: PASS, NO\_TEST, NO\_OPTION*

661 **R\_1** **IF** PCTS\_fdasync **THEN**  
662 **TEST:** A call to the *fdasync()* function returns an error that is detected.  
663 **NOTE:** There is no known portable test method for this assertion.  
664 **ELSE NO\_OPTION**  
665 **SEE:** Assertions in §6.6.2.4

666 **6** **IF** PCTS\_fdasync **THEN**  
667 **SETUP:** Open a file for read-write access without specifying O\_DSYNC, O\_RSYNC, or O\_DSYNC  
668 in the *oflags* argument of *open()*. Then write some data to the file and read from  
669 another place in the file.  
670 **TEST:** A call to the *fdasync()* function forces all currently queued I/O operations  
671 associated with the file indicated by file descriptor *fildes* to the synchronized I/O data  
672 integrity completion state. That is it completes by transferring an image of the data  
673 to the requesting process and any pending write requests affecting the data to be read  
674 are written to the physical medium containing the file prior to reading the data.  
675 **NOTE:** There is no known portable test method for this assertion.  
676 **ELSE NO\_OPTION**  
677 *Conformance for fdasync: PASS, NO\_TEST, NO\_OPTION*

678 **7** **IF** PCTS\_fdasync **THEN**  
679 **SETUP:** Open a file for read-write access without specifying O\_DSYNC, O\_RSYNC, or O\_DSYNC  
680 in the *oflags* argument of *open()*. Then write some data to the file.  
681 **TEST:** A call to the *fdasync()* function forces all currently queued I/O operations  
682 associated with the file indicated by file descriptor *fildes* to the synchronized I/O data  
683 integrity completion state. That is the data are written to the physical medium  
684 containing the file.  
685 **NOTE:** There is no known portable test method for this assertion.  
686 **ELSE NO\_OPTION**

- 687 *Conformance for fdatsync: PASS, NO\_TEST, NO\_OPTION*
- 688 **6.6.2.3 Returns**
- 689 **R\_2 IF** PCTS\_ *fdatsync* **THEN**  
 690 **TEST:** Upon successful completion, the *fdatsync()* function returns zero.  
 691 **ELSE** *NO\_OPTION*  
 692 **SEE:** Assertions in §6.6.2.2
- 693 **8 TEST:** Upon unsuccessful completion, the *fdatsync()* returns -1 and set *errno* to indicate the  
 694 error.  
 695 **SEE:** Assertions in §6.6.2.4  
 696 *Conformance for fdatsync: PASS*
- 697 **6.6.2.4 Errors**
- 698 **9 IF** PCTS\_ *fdatsync* **THEN**  
 699 **SETUP:** Call the *fdatsync()* function with the *fildev* argument not being a valid file  
 700 descriptor.  
 701 **TEST:** *fdatsync()* returns -1 and set *errno* to [EBADF].  
 702 **NOTE:** There is no known portable test method for this assertion.  
 703 **ELSE** *NO\_OPTION*  
 704 *Conformance for fdatsync: PASS, NO\_OPTION*
- 705 **10 IF** PCTS\_ *fdatsync* **THEN**  
 706 **IF** PCTS\_ *NO\_SYNC\_IO\_FILE* **THEN**  
 707 **SETUP:** Call the *fdatsync()* function with the *fildev* argument pointing to a file that  
 708 does not support synchronized I/O.  
 709 **TEST:** *fdatsync()* returns -1 and set *errno* to [EINVAL].  
 710 **ELSE** *NO\_TEST\_SUPPORT*  
 711 **ELSE** *NO\_OPTION*  
 712 *Conformance for fdatsync: PASS, NO\_OPTION*
- 713 **11 IF** not PCTS\_ *fdatsync* **THEN**  
 714 **SETUP:** Call the *fdatsync()* function even though it is not supported by this implementation  
 715 **TEST:** *fdatsync()* returns -1 and set *errno* to [ENOSYS].  
 716 **NOTE:** There is no known portable test method for this assertion.  
 717 **ELSE** *NO\_OPTION*  
 718 *Conformance for fdatsync: PASS, NO\_OPTION*
- 719 **12 IF** PCTS\_ *fdatsync* **THEN**  
 720 **TEST:** In the event that any of the queued I/O operations fail, *fdatsync()* returns the error  
 721 conditions defined for *read()* and *write()*.  
 722 **NOTE:** There is no known portable test method for this assertion.  
 723 **ELSE** *NO\_OPTION*  
 724 *Conformance for fdatsync: PASS, NO\_TEST, NO\_OPTION*

## 725 **6.7 Asynchronous Input and Output**

### 726 **6.7.1 Data Definitions for Asynchronous Input and Output**

727 There are no requirements for conforming implementations in this subclause.

#### 728 **6.7.1.1 Asynchronous I/O Control Block**

- 729 **1 SETUP:** Include the header `<aiocb.h>`  
 730 **TEST:** An asynchronous I/O control block structure *aio* is defined and has at least the following  
 731 members with the indicated types:

	<b>Member Type</b>	<b>Member Name</b>	<b>Description</b>
732	<i>int</i>	<i>aio_fildes</i>	File descriptor.
733	<i>off_t</i>	<i>aio_offset</i>	File offset.
734	<i>volatile void*</i>	<i>aio_buf</i>	Location of buffer.
735	<i>size_t</i>	<i>aio_nbytes</i>	Length of transfer.
736	<i>int</i>	<i>aio_reqprio</i>	Request priority offset.
737	<i>struct sigevent</i>	<i>aio_sigevent</i>	Signal number and value.
738	<i>int</i>	<i>aio_lio_opcode</i>	Operation to be performed.
739			
740			
741			<i>Conformance for aio.h: PASS</i>
742	<b>D-1</b>	<b>TEST:</b>	The PCD.1b documents any added extensions as permitted in 1.3.1.1 item (2) made to the <i>aioch</i> , including the definition of an environment in which an application can be run with the behavior specified by POSIX.1b {3}. in §6.7.1.1
743			
744			
745			<i>Conformance for aio.h: PASS</i>
746	<b>2</b>	<b>TEST:</b>	Added extensions to the <i>aioch</i> structure, are enabled as required by 1.3.1.1 that is, a Strictly Conforming $\pi 1$ Application does not need to be modified to execute in such a modified environment.
747			
748			
749		<b>NOTE:</b>	There is no known portable test method for this assertion.
750			<i>Conformance for aio.h: PASS, NO_TEST</i>
751	<b>3</b>	<b>FOR:</b>	<i>aio_read()</i> , <i>aio_write()</i> , and <i>aio_lio_listio()</i>
752		<b>IF</b>	<i>PCTS_function</i> <b>THEN</b>
753		<b>SETUP:</b>	Open a file so that <i>O_APPEND</i> is not set for the file descriptor <i>aio_fildes</i> , and <i>aio_fildes</i> is associated with a device that is capable of seeking
754		<b>TEST:</b>	The operation requested by <i>function()</i> takes place at the absolute position in the file as given by <i>aio_offset</i> , as if <i>lseek()</i> were called immediately prior to the operation with an <i>offset</i> argument equal to <i>aio_offset</i> and a <i>whence</i> argument equal to <i>SEEK_SET</i> .
755			
756			
757			
758		<b>ELSE</b>	<i>NO_OPTION</i>
759			<i>Conformance for aio.h: PASS, NO_OPTION</i>
760	<b>4</b>	<b>FOR:</b>	<i>aio_write()</i> , and <i>aio_lio_listio()</i>
761		<b>IF</b>	<i>PCTS_function</i> <b>THEN</b>
762		<b>IF</b>	<i>PCTS_APPEND_WRITE_SAME_ORDER</i> <b>THEN</b>
763		<b>SETUP:</b>	Open a file so that <i>O_APPEND</i> is set for the file descriptor. Open another file where <i>aio_fildes</i> is associated with a device that is incapable of seeking.
764		<b>TEST:</b>	Write operations for the <i>function()</i> function append to the file in the same order as the calls were made.
765			
766		<b>TR:</b>	Test for both files in the <i>SETUP</i> . Be sure to use the same priority for all writes
767			
768		<b>ELSE</b>	<i>NO_TEST_SUPPORT</i>
769		<b>ELSE</b>	<i>NO_OPTION</i>
770			<i>Conformance for aio.h: PASS, NO_TEST_SUPPORT, NO_OPTION</i>
771	<b>D_2</b>	<b>TEST:</b>	The PCD.1b documents under what circumstances the requirement to have write append operations occur in the same order as the calls may be relaxed in §6.7.1.1
772			
773			<i>Conformance for aio.h: PASS</i>
774	<b>5</b>	<b>FOR:</b>	<i>aio_read()</i> , <i>aio_write()</i> , and <i>aio_lio_listio()</i>
775		<b>IF</b>	<i>PCTS_function</i> and <i>{_POSIX_PRIORITIZED_IO}</i>
776			and <i>{_POSIX_PRIORITY_SCHEDULING}</i> <b>THEN</b>
777		<b>TEST:</b>	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.
778			
779		<b>ELSE</b>	<i>NO_OPTION</i>
780			<i>Conformance for aio.h: PASS, NO_OPTION</i>



- 781 **6** **FOR:** *aio\_read()*, *aio\_write()*, and *aio\_lio\_listio()*  
782 **IF** *PCTS\_function* and `{_POSIX_PRIORITIZED_IO}`  
783 and `{_POSIX_PRIORITY_SCHEDULING}` **THEN**  
784 **TEST:** The *aio\_reqprio* member can be used to lower, but not raise, the asynchronous I/O  
785 operation priority when it is within the range zero through `{AIO_PRIO_DELTA_MAX}`,  
786 inclusive.  
787 **ELSE NO\_OPTION**  
788 *Conformance for aio.h: PASS, NO\_OPTION*
- 789 **7** **FOR:** *aio\_read()*, *aio\_write()* and *aio\_lio\_listio()*  
790 **IF** *PCTS\_function* and `{POSIX_PRIORITY_SCHEDULING}` **THEN**  
791 **IF** *PCTS\_GTI\_DEVICE* **THEN**  
792 **TEST:** The priority of an asynchronous request is computed as process scheduling  
793 priority minus *aio\_reqprio*..  
794 **TR:** Test for a character special file.  
795 **ELSE NO\_TEST\_SUPPORT**  
796 **ELSE NO\_OPTION**  
797 *Conformance for aio.h: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 798 **8** **FOR:** *aio\_read()* *aio\_write*, and *aio\_lio\_listio()*  
799 **IF** *PCTS\_function* and `{POSIX_PRIORITIZED_IO}` **THEN**  
800 **IF** *PCTS\_GTI\_DEVICE* **THEN**  
801 **TEST:** Requests issued by *function()* with the same priority to a character special file  
802 are processed by the underlying device in FIFO order..  
803 **TR:** Test for both files in the SETUP. Be sure to use the same priority for all writes  
804 **ELSE NO\_TEST\_SUPPORT**  
805 **ELSE NO\_OPTION**  
806 *Conformance for aio.h: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 807 **D\_3** **IF** a PCD.1b documents the following **THEN**  
808 **TEST:** A PCD.1b that documents the order of processing of asynchronous I/O requests of the  
809 same priority issued to files that are not character special files does so in §6.7.1.1  
810 **ELSE NO\_OPTION**  
811 *Conformance for aio.h: PASS, NO\_OPTION*
- 812 **9** **FOR:** *aio\_read()* *aio\_write*, and *aio\_lio\_listio()*  
813 **IF** *PCTS\_function* and `{POSIX_PRIORITIZED_IO}` and  
814 `{POSIX_PRIORITY_SCHEDULING}` **THEN**  
815 **TEST:** The value of *aio\_reqprio* has no effect on process scheduling priority.  
816 **ELSE NO\_OPTION**  
817 *Conformance for aio.h: PASS, NO\_OPTION*
- 818 **10** **FOR:** *aio\_read()* *aio\_write*, and *aio\_lio\_listio()*  
819 **IF** *PCTS\_function* and `{POSIX_PRIORITIZED_IO}` and  
820 `{POSIX_PRIORITY_SCHEDULING}` **THEN**  
821 **SETUP:** Create prioritized asynchronous I/O requests to the same file that are blocked waiting  
822 for a resource required for that I/O operation.  
823 **TEST:** The higher-priority asynchronous I/O requests are granted the resource before  
824 lower-priority I/O requests are granted the resource.  
825 **ELSE NO\_OPTION**  
826 *Conformance for aio.h: PASS, NO\_OPTION*
- 827 **D\_4** **TEST:** The PCD.1b documents the relative priority of asynchronous I/O and synchronous I/O in §6.7.1.1.  
828 *Conformance for aio.h: PASS*
- 829 **11** **IF** `{_POSIX_PRIORITIZED_IO}` **THEN**  
830 **TEST:** The PCD.1b documents for which files I/O prioritization is supported in §6.7.1.1.  
831 **ELSE NO\_OPTION**  
832 *Conformance for aio.h: PASS, NO\_OPTION*

- 833 **12** **FOR:** *aio\_read()*, *aio\_write()*, and *aio\_lio\_listio()*  
834 **IF** *PCTS\_function* **THEN**  
835 **SETUP:** Use an *aio\_cb* where the *aio\_sigevent.sigev\_notify* member is SIGEV\_NONE  
836 **TEST:** After a call to the *function()* function, no signal is posted upon I/O completion, and  
837 the error status for the operation and the return status for the operation are set  
838 appropriately.  
839 **ELSE NO\_OPTION**  
840 *Conformance for aio.h: PASS, NO\_OPTION*
- 841 **13** **FOR:** *aio\_read()*, *aio\_write()*, and *aio\_lio\_listio()*  
842 **IF** *PCTS\_function* **THEN**  
843 **SETUP:** Use an *aio\_cb* where the *aio\_sigevent.sigev\_notify* member is SIGEV\_SIGNAL  
844 **TEST:** After a call to the *function()* function and upon I/O completion, the signal specified  
845 in *aio\_sigevent.sigev\_signo* is sent to the process.  
846 **ELSE NO\_OPTION**  
847 *Conformance for aio.h: PASS, NO\_OPTION*
- 848 **14** **FOR:** *aio\_read()*, *aio\_write()*, and *aio\_lio\_listio()*  
849 **IF** *PCTS\_function* {\_POSIX\_REALTIME\_SIGNALS} **THEN**  
850 **SETUP:** Use an *aio\_cb* where the *aio\_sigevent.sigev\_notify* member is SIGEV\_SIGNAL. Set the  
851 signal number to be queued to be in the range SIGRTMIN to SIGRTMAX and the set the  
852 SA\_SIGINFO flag for that signal number.  
853 **TEST:** After a call to the *function()* function and upon I/O completion, the signal will be  
854 queued to the process and the value specified in *aio\_sigevent.sigev\_value* will be the  
855 *si\_value* component of the generated signal.  
856 **ELSE NO\_OPTION**  
857 *Conformance for aio.h: PASS, NO\_OPTION*

858 NOTE: The following requirements of POSIX.1b {3} are explicitly required by the relevant functions and thus are  
859 not considered General Assertions and are to be tested in the functions with the requirements:

860 The return status of the asynchronous operation is the number of bytes transferred by the i/o operation. If the error  
861 status is set to indicate an error completion, then the return status is set to the return value that the corresponding  
862 *read()*, *write()*, or *fsynch()* call would have returned. When the error status is not equal to [EINPROGRESS], the return  
863 status shall reflect the return status of the corresponding synchronous operation.

#### 864 6.7.1.2 Manifest Constants

- 865 **15** **SETUP:** Include the header `<aio.h>`.  
866 **TEST:** The symbols AIO\_CANCELED, AIO\_NOTCANCELED, and AIO\_ALLDONE are defined and have  
867 unique values relative to each other.  
868 *Conformance for aio.h: PASS*
- 869 **16** **SETUP:** Include the header `<aio.h>`.  
870 **TEST:** The symbols LIO\_WAIT and LIO\_NOWAIT are defined and have unique values relative to each  
871 other.  
872 *Conformance for aio.h: PASS*
- 873 **17** **SETUP:** Include the header `<aio.h>`.  
874 **TEST:** The symbols LIO\_READ, LIO\_WRITE, and LIO\_NOP are defined and have unique values  
875 relative to each other.  
876 *Conformance for aio.h: PASS*

#### 877 6.7.2 Asynchronous Read

878 Function; *aio\_read()*

879 **6.7.2.1 Synopsis**880 **1**881 *M\_GA\_stdC\_proto\_decl(int; aio\_read; struct aiocb \*aiocbp; aio.h;;)*882 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3883 *Conformance for aio\_read: PASS[1, 2], NO\_OPTION*884 **2**885 *M\_GA\_commonC\_int\_result\_decl(aio\_read; aio.h;;)*886 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3887 *Conformance for aio\_read: PASS[1, 2], NO\_OPTION*888 **3**889 *M\_GA\_macro\_result\_decl(int; aio\_read; aio.h;;)*890 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4891 *Conformance for aio\_read: PASS, NO\_OPTION*892 **4**893 *M\_GA\_macro\_args(aio\_read; aio.h;;)*894 **SEE:** Assertion GA\_macro\_args in §2.7.3895 *Conformance for aio\_read: PASS, NO\_OPTION*896 **6.7.2.2 Description**897 **5****IF** *PCTS\_aio\_read* **THEN**898 **TEST:** A call to the *aio\_read()* function reads *aiocbp->aio\_nbytes* from the file associated  
899 with *aiocbp->aio\_fildes* into the buffer pointed to by *aiocbp->aio\_buf*.900 **ELSE** *NO\_OPTION*901 *Conformance for aio\_read: PASS, NO\_OPTION*902 **6****IF** *PCTS\_aio\_read* **THEN**903 **IF** *PCTS\_GTI\_DEVICE* **THEN**904 **TEST:** A call to the *aio\_read()* function returns when the read request has been  
905 initiated or queued to the file or device, even when the data cannot be delivered  
906 immediately.907 **TR:** Test for a character special device.908 **ELSE** *NO\_TEST\_SUPPORT*909 **ELSE** *NO\_OPTION*910 *Conformance for aio\_read: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*911 **7****IF** *PCTS\_aio\_read* and *{\_POSIX\_PRIORITIZED\_IO}* **THEN**912 **IF** *PCTS\_GTI\_DEVICE* **THEN**913 **SETUP:** Use a file for which prioritized I/O is supported.914 **TEST:** The asynchronous operation caused by the *aio\_read()* function is submitted at  
915 a priority equal to the scheduling priority of the process minus *aiocbp-*  
916 *>aio\_reqprio*.917 **NOTE:** There is no known portable test method for this assertion.918 **ELSE** *NO\_OPTION*919 *Conformance for aio\_read: PASS, NO\_OPTION*920 **8****IF** *PCTS\_aio\_read* and *{\_POSIX\_PRIORITIZED\_IO}* **THEN**921 **IF** *PCTS\_GTI\_DEVICE* **THEN**922 **SETUP:** Use a file for which prioritized I/O is supported.923 **TEST:** The asynchronous operation caused by the *aio\_read()* function is submitted at  
924 a priority equal to the scheduling priority of the process minus *aiocbp-*  
925 *>aio\_reqprio*.926 **TR:** Test for a character special file.927 **ELSE** *NO\_TEST\_SUPPORT*928 **ELSE** *NO\_OPTION*

929 *Conformance for aio\_read: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

930 **9** **IF** *PCTS\_aio\_read* and *PCTS\_aio\_error* **THEN**

931 **TEST:** The *aiocbp* value used in a call to the *aio\_read()* function when used as a *n* argument

932 to *aio\_error()* returns the error status of the asynchronous operation while it is

933 proceeding.

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

935 **ELSE** *NO\_OPTION*

936 *Conformance for aio\_read: PASS, NO\_TEST, NO\_OPTION*

937 **10** **IF** *PCTS\_aio\_read* and *PCTS\_aio\_error* **THEN**

938 **IF** *PCTS\_GTI\_DEVICE* **THEN**

939 **TEST:** The *aiocbp* value used in a call to the *aio\_read()* function when used as a *n*

940 argument to *aio\_error()* returns the error status of the asynchronous operation

941 while it is proceeding.

942 **TR:** Test for a character special file.

943 **ELSE** *NO\_TEST\_SUPPORT*

944 **ELSE** *NO\_OPTION*

945 *Conformance for aio\_read: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

946 **11** **IF** *PCTS\_aio\_read* and *PCTS\_aio\_return* **THEN**

947 **TEST:** The *aiocbp* value used in a call to the *aio\_read()* function when used as a *n* argument

948 to *aio\_return()* returns the return status of the asynchronous operation while it is

949 proceeding.

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

951 **ELSE** *NO\_OPTION*

952 *Conformance for aio\_read: PASS, NO\_TEST, NO\_OPTION*

953 **12** **IF** *PCTS\_aio\_read* and *PCTS\_aio\_return* **THEN**

954 **IF** *PCTS\_GTI\_DEVICE* **THEN**

955 **TEST:** The *aiocbp* value used in a call to the *aio\_read()* function when used as a *n*

956 argument to *aio\_return()* returns the return status of the asynchronous

957 operation while it is proceeding.

958 **TR:** Test for a character special file.

959 **ELSE** *NO\_TEST\_SUPPORT*

960 **ELSE** *NO\_OPTION*

961 *Conformance for aio\_read: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

962 **R\_1** **IF** *PCTS\_aio\_read* **THEN**

963 **TEST:** A call *aio\_read()* returns an error condition encountered during queuing without

964 having initiated or queued the request.

965 **ELSE** *NO\_OPTION*

966 **SEE:** Assertions in §6.7.2.4

967 **13** **IF** *PCTS\_aio\_read* **THEN**

968 **TEST:** A call *aio\_read()* causes the requested operation to take place at the absolute position

969 in the file as given by *aio\_offset*, as if *lseek()* were called immediately prior to the

970 operation with an *offset* equal to *aio\_offset* and a *whence* equal to *SEEK\_SET*.

971 **ELSE** *NO\_OPTION*

972 *Conformance for aio\_read: PASS, NO\_OPTION*

973 **D\_1** **IF** *PCTS\_aio\_read* and a PCD.1b documents the following **THEN**

974 **TEST:** A PCD.1b that documents the value of the file offset for the file after a successful call

975 to enqueue an asynchronous I/O operation does so in §6.7.2.2.

976 **ELSE** *NO\_OPTION*

977 *Conformance for aio\_read: PASS, NO\_OPTION*

978 **D\_2** **IF** *PCTS\_aio\_read* and a PCD.1b documents the following **THEN**

- 979                   **TEST:**     A PCD.1b that documents the behavior of a call to the *aio\_read()* when the buffer  
980                                   pointed to by *aiocbpl->aio\_buf* or the control block pointed to by *aiocbp* becomes  
981                                   an illegal address prior to asynchronous I/O completion does so in §6.7.2.2.  
982                   **ELSE NO\_OPTION**  
983                   *Conformance for aio\_read: PASS, NO\_OPTION*
- 984   **D\_3 IF** *PCTS\_aio\_read* and a PCD.1b documents the following **THEN**  
985                   **TEST:**     A PCD.1b that documents the results of simultaneous asynchronous operations using  
986                                   the same *aiocbp* does so in §6.7.2.2.  
987                   **ELSE NO\_OPTION**  
988                   *Conformance for aio\_read: PASS, NO\_OPTION*
- 989   **15 IF** *PCTS\_aio\_read* and {\_POSIX\_SYNCHRONIZED\_IO} **THEN**  
990                   **SETUP:**    Open a file by calling *open()* with O\_RSYNC and O\_DSYNC set in the *oflag* parameter.  
991                   **TEST:**     A read operation initiated by calling *aio\_read()* either completes by transferring an  
992                                   image of the data to the requesting process or, if unsuccessful, by diagnosing and  
993                                   returning an indicator of the error.  
994                   **TR:**     Test for regular files and, if PCTS\_GTI\_DEVICE, terminals  
995                   **NOTE:**    There is no known portable test method for this assertion.  
996                   **ELSE NO\_OPTION**  
997                   **SEE:**     Assertion GA\_syncIODataIntegrityRead in §2.2.2.119  
998                   *Conformance for aio\_read: PASS, NO\_TEST, NO\_OPTION*
- 999   **16 IF** *PCTS\_aio\_read* and {\_POSIX\_SYNCHRONIZED\_IO} **THEN**  
1000                   **SETUP:**    Open a file by calling *open()* with O\_RSYNC and O\_SYNC set in the *oflag* parameter.  
1001                   **TEST:**     At the time that the synchronized read operation initiated by calling *aio\_read()*  
1002                                   occurs, any pending write requests affecting the data to be read are written to the  
1003                                   physical medium containing the file prior to reading the data.  
1004                   **TR:**     Test for regular files.  
1005                   **NOTE:**    There is no known portable test method for this assertion.  
1006                   **ELSE NO\_OPTION**  
1007                   **SEE:**     Assertion GA\_syncIODataIntegrityWbeforeR in §2.2.2.119  
1008                   *Conformance for aio\_read: PASS, NO\_TEST, NO\_OPTION*
- 1009   **17 IF** *PCTS\_aio\_read* and {\_POSIX\_SYNCHRONIZED\_IO} **THEN**  
1010                   **SETUP:**    Open a file by calling *open()* with O\_RSYNC and O\_SYNC set in the *oflag* parameter.  
1011                   **TEST:**     At the time that the synchronized read operation initiated by calling *aio\_read()*  
1012                                   occurs, any pending write requests affecting the data to be read are written to the  
1013                                   physical medium containing the file prior to reading the data and the following file  
1014                                   attributes are also written to the physical medium containing the file prior to  
1015                                   returning to the calling process:
- 1016                                   1.   File mode.
  - 1017                                   2.   File serial number.
  - 1018                                   3.   ID of device containing this file.
  - 1019                                   4.   Number of links.
  - 1020                                   5.   User ID of the owner of the file.
  - 1021                                   6.   Group ID of the group of the file.
  - 1022                                   7.   The file size in bytes.
  - 1023                                   8.   Time of last access.
  - 1024                                   9.   Time of last data modification.

1025 10. Time of last file status change.  
 1026 **TR:** Test for regular files.  
 1027 **NOTE:** There is no known portable test method for this assertion.  
 1028 **ELSE NO\_OPTION**  
 1029 **SEE:** Assertion GA\_syncIOfileIntegrityRead in §2.2.2.120  
 1030 *Conformance for aio\_read: PASS, NO\_TEST, NO\_OPTION*

1031 **D\_4 IF** *PCTS\_aio\_read* and a PCD.1b documents the following **THEN**  
 1032 **TEST:** A PCD.1b that documents the result of any system action that changes the process  
 1033 memory space while an asynchronous I/O is outstanding to the address range being  
 1034 changed does so in §6.7.2.2.  
 1035 **ELSE NO\_OPTION**  
 1036 *Conformance for aio\_read: PASS, NO\_OPTION*

1037 **6.7.2.3 Returns**

1038 **R\_2 IF** *PCTS\_aio\_read* **THEN**  
 1039 **TEST:** The *aio\_read()* function returns the value zero to the calling process after the I/O  
 1040 operation is successfully queued.  
 1041 **ELSE NO\_OPTION**  
 1042 **SEE:** Assertions in §6.7.2.2.

1043 **R\_3 IF** *PCTS\_aio\_read* **THEN**  
 1044 **TEST:** The *aio\_read()* function returns the value -1 to the calling process and sets *errno* to  
 1045 indicate the error when the I/O operation not successfully queued.  
 1046 **ELSE NO\_OPTION**  
 1047 **SEE:** Assertions in §6.7.2.4.

1048 **6.7.2.4 Errors**

1049 **18 IF** *PCTS\_aio\_read* **THEN**  
 1050 **IF**  $\{AIO\_MAX\} \leq PCTS\_AIO\_MAX$  **THEN**  
 1051 **SETUP:** Queue  $\{AIO\_MAX\}$  asynchronous I/O operations before calling *aio\_read()*.  
 1052 **TEST:** A call to the *aio\_write()* function returns -1 and sets *errno* to [EAGAIN].  
 1053 **NOTE:** There is no known portable test method for this assertion.  
 1054 **ELSE NO\_TEST\_SUPPORT**  
 1055 **ELSE NO\_OPTION**  
 1056 *Conformance for aio\_read: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1057 **19 IF** *PCTS\_aio\_read* **THEN**  
 1058 **IF**  $\{AIO\_MAX\} \leq PCTS\_AIO\_MAX$  and *PCTS\_GTI\_DEVICE* **THEN**  
 1059 **SETUP:** Queue  $\{AIO\_MAX\}$  asynchronous I/O operations before calling *aio\_read()*.  
 1060 **TEST:** A call to the *aio\_read()* function returns -1 and sets *errno* to [EAGAIN].  
 1061 **TR:** Test for a character special device.  
 1062 **ELSE NO\_TEST\_SUPPORT**  
 1063 **ELSE NO\_OPTION**  
 1064 *Conformance for aio\_read: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1065 **20 IF** *PCTS\_aio\_read* **THEN**  
 1066 **IF**  $\{AIO\_MAX\} > PCTS\_AIO\_MAX$  and *PCTS\_GTI\_DEVICE* **THEN**  
 1067 **SETUP:** Queue  $PCTS\_AIO\_MAX - 1$  asynchronous I/O operations before calling *aio\_read()*.  
 1068 **TEST:** A call to the *aio\_read()* function returns 0 and does not set *errno* to [EAGAIN].  
 1069 **TR:** Test for a character special device.  
 1070 **ELSE NO\_TEST\_SUPPORT**  
 1071 **ELSE NO\_OPTION**  
 1072 *Conformance for aio\_read: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1073     **21**     **IF** not *PCTS\_aio\_read* **THEN**  
1074             **TEST:**     A call to the *aio\_read()* function returns -1 and sets *errno* to [ENOSYS].  
1075             **ELSE NO\_OPTION**  
1076             Conformance for *aio\_read*: PASS, NO\_OPTION

1077     **ebadf1**   **IF** *PCTS\_aio\_read* **THEN**  
1078             **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
1079                 **TEST:**     After a call to the *aio\_read()* function where the *aioctx->aio\_fildes* argument  
1080                             is not a valid file descriptor, the implementation either detects the condition  
1081                             synchronously and then the *aio\_read()* function returns -1 and sets *errno* to  
1082                             [EBADF] or it detects the condition asynchronously and then the return status  
1083                             of the asynchronous operation is set to -1, and the error status of the  
1084                             asynchronous operation is set to [EBADF].  
1085                 **ELSE NO\_TEST\_SUPPORT**  
1086             **ELSE NO\_OPTION**  
1087             Conformance for *aio\_read*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1088     **ebadf2**   **IF** *PCTS\_aio\_read* **THEN**  
1089             **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
1090                 **TEST:**     After a call to the *aio\_read()* function where the *aioctx->aio\_fildes* argument  
1091                             is not open for reading, the implementation either detects the condition  
1092                             synchronously and then the *aio\_read()* function returns -1 and sets *errno* to  
1093                             [EBADF] or it detects the condition asynchronously and then the return status  
1094                             for the asynchronous operation is set to -1, and the error status of the  
1095                             asynchronous operation is set to [EBADF].  
1096                 **ELSE NO\_TEST\_SUPPORT**  
1097             **ELSE NO\_OPTION**  
1098             Conformance for *aio\_read*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1099     **EINVAL1**   **IF** *PCTS\_aio\_read* **THEN**  
1100             **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
1101                 **TEST:**     After a call to the *aio\_read()* function where the file offset value implied by  
1102                             *aioctx->aio\_offset* would be invalid, *aioctx->aio\_reqprio* is not a valid value,  
1103                             or *aioctx->aio\_nbytes* is an invalid value, the implementation either detects  
1104                             the condition synchronously and then the *aio\_read()* function returns -1 and  
1105                             sets *errno* to [EINVAL] or it detects the condition asynchronously and then the  
1106                             return status of the asynchronous operation is set to [EINVAL].  
1107                 **TR:**     Test separately for each of the three conditions above.  
1108                 **ELSE NO\_TEST\_SUPPORT**  
1109             **ELSE NO\_OPTION**  
1110             Conformance for *aio\_read*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1111     **26**     **IF** *PCTS\_aio\_read* **THEN**  
1112             **IF** *PCTS\_aio\_cancel* and *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
1113                 **SETUP:**   Call the *aio\_read()* function so that it successfully queues the I/O operation.  
1114                 **TEST:**     After a read operation is canceled by a call to the *aio\_cancel()* function, the  
1115                             return status of the asynchronous operation returned by a call to the  
1116                             *aio\_return()* function is -1 and the error status returned by a call to the  
1117                             *aio\_error()* function is [EINTR] or [ECANCELED].  
1118                 **TR:**     Test for a pipe and a FIFO.

1119                             If *PCTS\_GTI\_DEVICE*, test for a character special file.  
1120                 **ELSE NO\_TEST\_SUPPORT**  
1121             **ELSE NO\_OPTION**  
1122             Conformance for *aio\_read*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1123 **R\_4 IF** *PCTS\_aio\_read* **THEN**  
 1124     **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
 1125         **TEST:** After a call to the *aio\_read()* function where the *aio\_bp->aio\_fildes* argument  
 1126                 is not a valid file descriptor, the implementation either detects the condition  
 1127                 synchronously and then the *aio\_read()* function returns -1 and sets *errno* to  
 1128                 [EBADF] or it detects the condition asynchronously and then the return status  
 1129                 of the asynchronous operation is set to -1, and the value of the call to  
 1130                 *aio\_error()* is set to [EBADF].  
 1131         **ELSE NO\_TEST\_SUPPORT**  
 1132     **ELSE NO\_OPTION**  
 1133     **SEE:** Assertion ebadf1 in §6.7.2.4.

1134 **R\_5 IF** *PCTS\_aio\_read* **THEN**  
 1135     **IF** *PCTS\_aio\_error* **THEN**  
 1136         **TEST:** After a call to the *aio\_read()* function where the *aio\_bp->aio\_fildes* argument  
 1137                 is not open for reading, the implementation either detects the condition  
 1138                 asynchronously and then the *aio\_read()* function returns -1 and sets *errno* to  
 1139                 [EBADF] or it detects the condition asynchronously and then the return status  
 1140                 of the asynchronous operation is set to -1, and the error status of the  
 1141                 asynchronous operation is set to [EBADF].  
 1142         **ELSE NO\_TEST\_SUPPORT**  
 1143     **ELSE NO\_OPTION**  
 1144     **SEE:** Assertion ebadf1 in §6.7.2.4.

1145 **29 IF** *PCTS\_aio\_read* and *PCTS\_aio\_cancel* **THEN**  
 1146     **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
 1147         **SETUP:** Call the *aio\_read()* function so that it successfully queues the I/O operation.  
 1148         **TEST:** A read operation that is canceled after a call to the *aio\_cancel()* function and  
 1149                 before the I/O completed results in a return status returned by a call to the  
 1150                 *aio\_return()* function that is -1 and the error status returned by a call to the  
 1151                 *aio\_error()* function that is [ECANCELED].  
 1152         **TR:** Test for test for a pipe and a FIFO.  
  
 1153                 If *PCTS\_GTI\_DEVICE*, test for a character special file.  
 1154         **ELSE NO\_TEST\_SUPPORT**  
 1155     **ELSE NO\_OPTION**  
 1156     *Conformance for aio\_read: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1157 **R\_6 IF** *PCTS\_aio\_read* **THEN**  
 1158     **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
 1159         **TEST:** After a call to the *aio\_read()* function where the file offset value implied by  
 1160                 *aio\_bp->aio\_offset* would be invalid, the implementation detects the condition  
 1161                 asynchronously and then the return status of the asynchronous operation is set  
 1162                 to -1, and the value of the call to *aio\_error()* is set to [EBADF].  
 1163         **ELSE NO\_TEST\_SUPPORT**  
 1164     **ELSE NO\_OPTION**  
 1165     **SEE:** Assertion einval in §6.7.2.4.

### 1166 6.7.3 Asynchronous Write

1167 Function: *aio\_write()*

#### 1168 6.7.3.1 Synopsis

1169 **1**  
 1170 *M\_GA\_stdC\_proto\_decl(int; aio\_write; struct aiocb \* aio\_bp; aio.h;;)*  
 1171 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 1172 *Conformance for aio\_write: PASS[1, 2], NO\_OPTION*



- 1173     **2**  
 1174     *M\_GA\_commonC\_int\_result\_decl(aio\_write; aio.h;;;)*  
 1175     **SEE:**     Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 1176     *Conformance for aio\_write: PASS[1, 2], NO\_OPTION*
- 1177     **3**  
 1178     *M\_GA\_macro\_result\_decl(int; aio\_write; aio.h;;;)*  
 1179     **SEE:**     Assertion GA\_macro\_result\_decl in §1.3.4  
 1180     *Conformance for aio\_write: PASS[1, 2], NO\_OPTION*
- 1181     **4**  
 1182     *M\_GA\_macro\_args ( aio\_write; aio.h;;;)*  
 1183     **SEE:**     Assertion GA\_macro\_args in §2.7.3  
 1184     *Conformance for aio\_write: PASS, NO\_OPTION*
- 1185     **6.7.3.2 Description**
- 1186     **5**     **IF** *PCTS\_aio\_write* **THEN**  
 1187         **TEST:**     A call to the *aio\_write()* function writes *aiocbp->aio\_nbytes* to the file associated  
 1188                     with *aiocbp->aio\_fildes* from the buffer pointed to by *aiocbp->aio\_buf*.  
 1189     **ELSE** *NO\_OPTION*  
 1190     *Conformance for aio\_write: PASS, NO\_OPTION*
- 1191     **6**     **IF** *PCTS\_aio\_write* **THEN**  
 1192         **IF** *PCTS\_GTI\_DEVICE* **THEN**  
 1193             **TEST:**     A call to the *aio\_write()* function returns when the write request has been  
 1194                     initiated or queued to the file or device.  
 1195             **TR:**     Test for a character special device.  
 1196         **ELSE** *NO\_TEST\_SUPPORT*  
 1197     **ELSE** *NO\_OPTION*  
 1198     *Conformance for aio\_write: PASS,NO\_TEST\_SUPPORT, NO\_OPTION*
- 1199     **7**     **IF** *PCTS\_aio\_write* and {*\_POSIX\_PRIORITIZED\_IO*} **THEN**  
 1200         **SETUP:**     Use a file for which prioritized I/O is supported.  
 1201         **TEST:**     The asynchronous operation caused by the *aio\_write()* function is submitted at a  
 1202                     priority equal to the scheduling priority of the process minus *aiocbp->aio\_reqprio*.  
 1203         **NOTE:**     There is no known portable test method for this assertion.
- 1204     **ELSE** *NO\_OPTION*  
 1205     *Conformance for aio\_write: PASS,NO\_TEST\_SUPPORT, NO\_OPTION*
- 1206     **8**     **IF** *PCTS\_aio\_write* and {*\_POSIX\_PRIORITIZED\_IO*} **THEN**  
 1207         **IF** *PCTS\_GTI\_DEVICE* **THEN**  
 1208             **SETUP:**     Use a file for which prioritized I/O is supported.  
 1209             **TEST:**     The asynchronous operation caused by the *aio\_write()* function is submitted at  
 1210                     a priority equal to the scheduling priority of the process minus *aiocbp-*  
 1211                     *>aio\_reqprio*.  
 1212             **TR:**     Test for a character special file.  
 1213             **NOTE:**     There is no known portable test method for this assertion.  
 1214         **ELSE** *NO\_TEST\_SUPPORT*  
 1215     **ELSE** *NO\_OPTION*  
 1216     *Conformance for aio\_write: PASS,NO\_TEST\_SUPPORT, NO\_OPTION*
- 1217     **9**     **IF** *PCTS\_aio\_write* and *PCTS\_aio\_error* **THEN**  
 1218         **TEST:**     The *aiocbp* value used in a call to the *aio\_write()* function when used as an argument  
 1219                     to *aio\_error()* returns the error status of the asynchronous operation while it is  
 1220                     proceeding.  
 1221         **NOTE:**     There is no known portable test method for this assertion.

1222           **ELSE NO\_OPTION**  
1223           *Conformance for aio\_write: PASS, NO\_TEST, NO\_OPTION*

1224    **10**    **IF PCTS\_aio\_write and PCTS\_aio\_error THEN**  
1225            **IF PCTS\_GTI\_DEVICE THEN**  
1226                **TEST:**    The *aiocbp* value used in a call to the *aio\_write()* function when used as an  
1227                                argument to *aio\_error()* returns the error status of the asynchronous operation  
1228                                while it is proceeding.  
1229                **TR:**    Test for a character special file.  
1230                **ELSE NO\_TEST\_SUPPORT**  
1231            **ELSE NO\_OPTION**  
1232            *Conformance for aio\_write: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1233    **11**    **IF PCTS\_aio\_write and PCTS\_aio\_return THEN**  
1234                **TEST:**    The *aiocbp* value used in a call to the *aio\_write()* function when used as an argument  
1235                                to *aio\_return()* returns the error status of the asynchronous operation while it is  
1236                                proceeding.  
1237                **NOTE:**    There is no known portable test method for this assertion.  
1238                **ELSE NO\_OPTION**  
1239                *Conformance for aio\_write: PASS, NO\_TEST, NO\_OPTION*

1240    **12**    **IF PCTS\_aio\_write and PCTS\_aio\_return THEN**  
1241                **IF PCTS\_GTI\_DEVICE THEN**  
1242                **TEST:**    The *aiocbp* value used in a call to the *aio\_write()* function when used as an  
1243                                argument to *aio\_return()* returns the return status of the asynchronous operation  
1244                                while it is proceeding.  
1245                **TR:**    Test for a character special file.  
1246                **ELSE NO\_TEST\_SUPPORT**  
1247                **ELSE NO\_OPTION**  
1248                *Conformance for aio\_write: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1249    **D\_1** **IF PCTS\_aio\_write and a PCD.1b documents the following THEN**  
1250                **TEST:**    A PCD.1b that documents the behavior of a call to the *aio\_write()* when the buffer  
1251                                pointed to by *aiocbp->aio\_buf* or the control block pointed to by *aiocbp* becomes an  
1252                                illegal process prior to asynchronous I/O completion does so in §6.7.3.2.  
1253                **ELSE NO\_OPTION**  
1254                *Conformance for aio\_write: PASS, NO\_OPTION*

1255    **13**    **IF PCTS\_aio\_write THEN**  
1256                **SETUP:**    Use a file where the O\_APPEND flag is not set for the file descriptor *aio\_fildes*.  
1257                **TEST:**    A call *aio\_write()* causes the requested operation to take place at the absolute position  
1258                                in the file as given by *aio\_offset*, as if *lseek()* were called immediately prior to the  
1259                                operation with an *offset* equal to *aio\_offset* and a *whence* equal to SEEK\_SET.  
1260                **ELSE NO\_OPTION**  
1261                *Conformance for aio\_write: PASS, NO\_OPTION*

1262    **14**    **IF PCTS\_aio\_write THEN**  
1263                **SETUP:**    Use a file where the O\_APPEND flag is set for the file descriptor.  
1264                **TEST:**    Write operations caused by calling the *aio\_write()* function append to the file in the  
1265                                same order as the calls were made.  
1266                **ELSE NO\_OPTION**  
1267                *Conformance for aio\_write: PASS, NO\_OPTION*

1268    **D\_2** **IF PCTS\_aio\_write and a PCD.1b documents the following THEN**  
1269                **TEST:**    A PCD.1b that documents the value of the file offset for the file after a successful call  
1270                                to enqueue an asynchronous I/O operation does so in §6.7.3.2.  
1271                **ELSE NO\_OPTION**  
1272                *Conformance for aio\_write: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

- 1273 **15** **IF** *PCTS\_aio\_write* **THEN**  
 1274 **TEST:** The *aioebp->aio\_lio\_opcode* field is ignored by *aio\_write()*.  
 1275 **ELSE NO\_OPTION**  
 1276 *Conformance for aio\_write: PASS, NO\_OPTION*
- 1277 **D\_3** **IF** *PCTS\_aio\_write* and a PCD.1b documents the following **THEN**  
 1278 **TEST:** A PCD.1b that documents the results of simultaneous asynchronous operations using  
 1279 the same *aioebp* does so in §6.7.3.2.  
 1280 **ELSE NO\_TEST\_SUPPORT**  
 1281 **ELSE NO\_OPTION**  
 1282 *Conformance for aio\_write: PASS, NO\_OPTION*
- 1283 **16** **IF** *PCTS\_aio\_write* and `{_POSIX_SYNCHRONIZED_IO}` **THEN**  
 1284 **SETUP:** Open a file by calling *open()* with `O_DSYNC` set in the *oflag* parameter.  
 1285 **TEST:** A write operation initiated by calling *aio\_write()* either completes by transferring an  
 1286 image of the data to the physical medium containing the file or, if unsuccessful, by  
 1287 diagnosing and returning an indicator of the error.  
 1288 **TR:** Test for regular files, and, if `PCTS_GTI_DEVICE`, terminals.  
 1289 **NOTE:** There is no known portable test method for this assertion.  
 1290 **ELSE NO\_OPTION**  
 1291 **SEE:** Assertion `GA_syncIODataIntegrityWrite` in §2.2.2.119  
 1292 *Conformance for aio\_write: PASS, NO\_TEST, NO\_OPTION*
- 1293 **17** **IF** *PCTS\_aio\_write* and `{_POSIX_SYNCHRONIZED_IO}` **THEN** *PCTS\_aio\_write* and  
 1294 `{_POSIX_SYNCHRONIZED_IO}`  
 1295 **SETUP:** Open a file by calling *open()* with `O_SYNC` set in the *oflag* parameter.  
 1296 **TEST:** At a time that the synchronized write operation initiated by calling *aio\_write()* occurs,  
 1297 the data are written to the physical medium containing the file and the following file  
 1298 attributes are also written to the physical medium containing the file prior to returning  
 1299 to the calling process:
- 1300 1. File mode.
  - 1301 2. File serial number.
  - 1302 3. ID of device containing this file.
  - 1303 4. Number of links.
  - 1304 5. User ID of the owner of the file.
  - 1305 6. Group ID of the group of the file.
  - 1306 7. The file size in bytes.
  - 1307 8. Time of last access.
  - 1308 9. Time of last data modification.
  - 1309 10. Time of last file status change.
- 1310 **TR:** Test for regular files.  
 1311 **NOTE:** There is no known portable test method for this assertion.  
 1312 **ELSE NO\_OPTION**  
 1313 **SEE:** Assertion `GA_syncIOFileIntegrityWrite` in §2.2.2.120  
 1314 *Conformance for aio\_write: PASS, NO\_TEST, NO\_OPTION*
- 1315 **D\_4** **IF** *PCTS\_aio\_write* and a PCD.1b documents the following **THEN**

1316                   **TEST:**     A PCD.1b that documents the result of any system action that changes the process  
 1317                                   memory space while an asynchronous I/O is outstanding to the address range being  
 1318                                   changed does so in §6.7.3.2.  
 1319                   **ELSE NO\_OPTION**  
 1320                   *Conformance for aio\_write: PASS, NO\_OPTION*

### 1321     **6.7.3.3 Returns**

1322     **R\_1 IF PCTS\_aio\_write THEN**  
 1323                   **TEST:**     The *aio\_write()* function returns the value zero to the calling process after the I/O  
 1324                                   operation is successfully queued.  
 1325                   **ELSE NO\_OPTION**  
 1326                   **SEE:**        Assertions in §6.7.3.2.

1327     **R\_2 IF PCTS\_aio\_write THEN**  
 1328                   **TEST:**     The *aio\_write()* function returns the value of -1 to the calling process and sets *errno*  
 1329                                   to indicate the error when the I/O operation is not successfully queued.  
 1330                   **ELSE NO\_OPTION**  
 1331                   **SEE:**        Assertions in §6.7.3.4.

### 1332     **6.7.3.4 Errors**

1333     **18     IF PCTS\_aio\_write THEN**  
 1334                   **IF {AIO\_MAX} ≤ PCTS\_AIO\_MAX THEN**  
 1335                                   **SETUP:**   Queue {AIO\_MAX} asynchronous I/O operations before calling *aio\_write()*.  
 1336                                   **TEST:**     A call to the *aio\_write()* function returns -1 and sets *errno* to [EAGAIN].  
 1337                                   **NOTE:**     There is no known portable test method for this assertion.  
 1338                                   **ELSE NO\_TEST\_SUPPORT**  
 1339                   **ELSE NO\_OPTION**  
 1340                   *Conformance for aio\_write: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1341     **19     IF PCTS\_aio\_write THEN**  
 1342                   **IF {AIO\_MAX} ≤ PCTS\_AIO\_MAX and THEN**  
 1343                                   **SETUP:**   Queue {AIO\_MAX} asynchronous I/O operations before calling *aio\_write()*.  
 1344                                   **TEST:**     A call to the *aio\_write* function returns -1 and sets *errno* to [EAGAIN]  
 1345                                   **TR:**        Test for a character special device.  
 1346                                   **ELSE NO\_TEST\_SUPPORT**  
 1347                   **ELSE NO\_OPTION**  
 1348                   *Conformance for aio\_write: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1349     **20     IF PCTS\_aio\_write THEN**  
 1350                   **IF {AIO\_MAX} PCTS\_AIO\_MAX and PCTS\_GTI\_DEVICE THEN**  
 1351                                   **SETUP:**   Queue *PCTS\_AIO\_MAX* asynchronous I/O operations before calling *aio\_write()*.  
 1352                                   **TEST:**     A call to the *aio\_write()* function returns 0 and does not set *errno* to [EAGAIN].  
 1353                                   **TR:**        Test for a character special device.  
 1354                                   **ELSE NO\_TEST\_SUPPORT**  
 1355                   **ELSE NO\_OPTION**  
 1356                   *Conformance for aio\_write: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1357     **21     IF not PCTS\_aio\_write THEN**  
 1358                   **TEST:**     A call to the *aio\_write()* function returns -1 and sets *errno* to [ENOSYS].  
 1359                   **ELSE NO\_OPTION**  
 1360                   *Conformance for aio\_write: PASS, NO\_OPTION*

1361     **ebadf1   IF PCTS\_aio\_write THEN**  
 1362                   **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**

1363                   **TEST:**     After a call to the *aio\_write()* function where the *aioctx->aio\_fildes* argument  
1364                   is not a valid file descriptor, the implementation either detects the condition  
1365                   synchronously and then the *aio\_write()* function returns -1 and sets *errno* to  
1366                   [EBADF] or it detects the condition asynchronously and then the return status of  
1367                   the asynchronous operation is set to -1, and the error status of the asynchronous  
1368                   operation is set to [EBADF].  
1369                   **ELSE NO\_TEST\_SUPPORT**  
1370                   **ELSE NO\_OPTION**  
1371                   Conformance for *aio\_write*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1372   **ebadf2**   **IF PCTS\_aio\_write THEN**  
1373                   **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**  
1374                   **TEST:**     After a call to the *aio\_write()* function where the *aioctx->aio\_fildes* argument  
1375                   is not open for writing, the implementation either detects the condition  
1376                   synchronously and then the *aio\_write()* function returns -1 and sets *errno* to  
1377                   [EBADF] or it detects the condition asynchronously and then the return status of  
1378                   the asynchronous operation is set to -1, and the error status of the asynchronous  
1379                   operation is set to [EBADF].  
1380                   **ELSE NO\_TEST\_SUPPORT**  
1381                   **ELSE NO\_OPTION**  
1382                   Conformance for *aio\_write*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1383   **EINVAL1** **IF PCTS\_aio\_write THEN**  
1384                   **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**  
1385                   **TEST:**     After a call to the *aio\_write()* function where the file offset value implied by  
1386                   *aioctx->aio\_offset* would be invalid, *aioctx->aio\_reqprio* is not a valid value,  
1387                   or *aioctx->aio\_nbytes* is an invalid value, the implementation either detects the  
1388                   condition synchronously and then the *aio\_write()* function returns -1 and sets  
1389                   *errno* to [EINVAL] or it detects the condition asynchronously and then the return  
1390                   status of the asynchronous operation is set to -1, and the error status of the  
1391                   asynchronous operation is set to [EINVAL]  
1392                   **TR:** Test separately for each of the three conditions above.  
1393                   **ELSE NO\_TEST\_SUPPORT**  
1394                   **ELSE NO\_OPTION**  
1395                   Conformance for *aio\_write*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1396   **22**       **IF PCTS\_aio\_write THEN**  
1397                   **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**  
1398                   **SETUP:**   Call the *aio\_write()* function so that it successfully queues the I/O operation for  
1399                   a pipe or FIFO that has the O\_NONBLOCK flag set for the file descriptor and with  
1400                   the *aioctx->aio\_nbytes* is less than or equal to {PIPE\_BUF} and where there is  
1401                   insufficient capacity to accept the data.  
1402                   **TEST:**     The return status of the asynchronous operation returned by a call to the  
1403                   *aio\_return()* function is -1 and the error status returned by a call to the  
1404                   *aio\_error()* function is [EAGAIN].  
1405                   **TR:** Test for both a pipe and FIFO. If {PIPE\_BUF} is greater than *PCTS\_PIPE\_BUF*, test with  
1406                   values of *aioctx->aio\_nbytes* up to and including *PCTS\_PIPE\_BUF*.  
1407                   **ELSE NO\_TEST\_SUPPORT**  
1408                   **ELSE NO\_OPTION**  
1409                   Conformance for *aio\_write*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1410   **24**       **IF PCTS\_aio\_write THEN**  
1411                   **IF PCTS\_aio\_cancel and PCTS\_aio\_error and PCTS\_aio\_return THEN**  
1412                   **SETUP:**   Call the *aio\_write()* function so that it successfully queues the I/O operation.  
1413                   **TEST:**     After a write operation is canceled by a call to the *aio\_cancel()* function and no  
1414                   data were transferred, the return status of the asynchronous operation returned  
1415                   by a call to the *aio\_return()* function is -1 and the error status returned by a call  
1416                   to the *aio\_error()* function is [EINTR] or [ECANCELED].

1417 **TR:** Test for test for a pipe and a FIFO.

1418 If *PCTS\_GTI\_DEVICE*, test for a character special file.

1419 **ELSE NO\_TEST\_SUPPORT**

1420 **ELSE NO\_OPTION**

1421 *Conformance for aio\_write: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1422 **25 IF PCTS\_aio\_write THEN**

1423 **IF** the implementation supports a maximum file size **THEN**

1424 **TEST:** An attempt to write a file that would exceed an implementation-defined

1425 maximum size by calling the *aio\_write()* function and where the write operation

1426 is successfully queued, causes the error status for the asynchronous operation to

1427 be [EFBIG].

1428 **NOTE:** The assertion test would require an unreasonable amount of time or resources

1429 on most implementations.

1430 **ELSE NO\_TEST\_SUPPORT**

1431 **ELSE NO\_OPTION**

1432 *Conformance for aio\_write: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1433 **26 IF PCTS\_aio\_write and {\_POSIX\_JOB\_CONTROL} THEN**

1434 **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**

1435 **SETUP:** Make the testing process part of the background process group, set TOSTOP,

1436 ignore or block SIGTTOU signals, and make the process group of the process

1437 orphaned. Call the *aio\_write()* function so that it successfully queues the I/O

1438 operation to write from its controlling terminal.

1439 **TEST:** The write operation will fail and the return status of the asynchronous operation

1440 returned by a call to the *aio\_return()* function is -1 and the error status returned

1441 by a call to the *aio\_error()* function is [EIO].

1442 **TR:** Test for test for a pipe and a FIFO.

1443 **ELSE NO\_TEST\_SUPPORT**

1444 **ELSE NO\_OPTION**

1445 *Conformance for aio\_write: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1446 **27 IF PCTS\_aio\_write THEN**

1447 **IF PCTS\_aio\_error and PCTS\_aio\_return and PCTS\_DETECT\_ENOSPC THEN**

1448 **SETUP:** Fill a device so that there is no more space available on it for data. Call the

1449 *aio\_write()* function so that it successfully queues the I/O operation.

1450 **TEST:** After a call to the *aio\_write()* function, the implementation either detects the

1451 condition synchronously and the *aio\_write()* function returns -1 and sets *errno*

1452 to [EBADF] or it detects the condition asynchronously and the return status of the

1453 asynchronous operation is set to -1, and the error status of the asynchronous

1454 operation is set to [EBADF].

1455 **ELSE NO\_TEST\_SUPPORT**

1456 **ELSE NO\_OPTION**

1457 *Conformance for aio\_write: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1458 **28 IF PCTS\_aio\_write THEN**

1459 **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**

1460 **SETUP:** Call the *aio\_write()* function so that it successfully queues the I/O operation.

1461 **TEST:** After a call to the *aio\_write()* function to write to a pipe or FIFO that is not open

1462 for reading for any process, the implementation either detects the condition

1463 synchronously and the *aio\_write()* function returns -1 and sets *errno* to [EBADF]

1464 or it detects the condition asynchronously and the return status of the

1465 asynchronous operation is set to -1, and the error status of the asynchronous

1466 operation is set to [EBADF].

1467 **ELSE NO\_TEST\_SUPPORT**

1468 **ELSE NO\_OPTION**

1469 *Conformance for aio\_write: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1470 **R\_3 IF PCTS\_aio\_write THEN**  
 1471 **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**  
 1472 **TEST:** After a call to the *aio\_write()* function where the *aioebp->aio\_fildes* argument  
 1473 is not a valid file descriptor, the implementation either detects the condition  
 1474 asynchronously and the *aio\_write()* function returns -1 and sets *errno* to  
 1475 [EBADF] or it detects the condition asynchronously and the error status of the  
 1476 asynchronous operation is set to -1, and the error status of the asynchronous  
 1477 operation is set to [EBADF].  
 1478 **ELSE NO\_TEST\_SUPPORT**  
 1479 **ELSE NO\_OPTION**  
 1480 **SEE:** Assertion ebadf1 in §6.7.3.4.

1481 **R\_4 IF PCTS\_aio\_write THEN**  
 1482 **IF PCTS\_aio\_error THEN**  
 1483 **TEST:** After a call to the *aio\_write()* function where the *aioebp->aio\_fildes* argument  
 1484 is not open for writing, the implementation either detects the condition  
 1485 synchronously and then the *aio\_write()* function returns -1 and sets *errno* to  
 1486 [EBADF] or it detects the condition asynchronously and then the return status of  
 1487 the asynchronous operation is set to -1, and the error status of the asynchronous  
 1488 operation is set to [EBADF].  
 1489 **ELSE NO\_TEST\_SUPPORT**  
 1490 **ELSE NO\_OPTION**  
 1491 **SEE:** Assertion ebadf1 in §6.7.3.4.

1492 **29 IF PCTS\_aio\_write and PCTS\_aio\_cancel THEN**  
 1493 **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**  
 1494 **SETUP:** Call the *aio\_write()* function so that it successfully queues the I/O operation.  
 1495 **TEST:** A write operation that is canceled after call to the *aio\_cancel()* function and  
 1496 before the I/O completed results in a return status returned by a call to the  
 1497 *aio\_return()* function that is -1 and the error status returned by a call to the  
 1498 *aio\_error()* function that is [ECANCELED].  
 1499 **TR:** Test for test for a pipe and a FIFO.

1500 If *PCTS\_GTI\_DEVICE*, test for a character special file.  
 1501 **ELSE NO\_TEST\_SUPPORT**  
 1502 **ELSE NO\_OPTION**  
 1503 *Conformance for aio\_write: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1504 **R\_5 IF PCTS\_aio\_write THEN**  
 1505 **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**  
 1506 **TEST:** After a call to the *aio\_write()* function where file offset value implied by  
 1507 *aioebp->aio\_offset* would be invalid, the implementation detects the condition  
 1508 asynchronously and then the return status of the asynchronous operation is set  
 1509 to -1, and the error status of the asynchronous operation is set to [EINVAL].  
 1510 **ELSE NO\_TEST\_SUPPORT**  
 1511 **ELSE NO\_OPTION**  
 1512 **SEE:** Assertion einval in §6.7.3.4.

## 1513 6.7.4 List Directed I/O

1514 Function *lio\_listio()*

### 1515 6.7.4.1 Synopsis

1516 **1**  
 1517 *M\_GA\_stdC\_proto\_decl(int; lio\_listio; int mode, struct aiocb \*const list[], int nent, struct sigevent*  
 1518 *\*sig; aio.h;);*  
 1519 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3

- 1520 *Conformance for lio\_listio: PASS[1, 2], NO\_OPTION*
- 1521 **2**
- 1522 *M\_GA\_commonC\_int\_result\_decl(lio\_listio; aio.h;;;)*
- 1523 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3
- 1524 *Conformance for lio\_listio: PASS[1, 2], NO\_OPTION*
- 1525 **3**
- 1526 *M\_GA\_macro\_result\_decl(int; lio\_listio; aio.h;;;)*
- 1527 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4
- 1528 *Conformance for lio\_listio: PASS, NO\_OPTION*
- 1529 **4**
- 1530 *M\_GA\_macro\_args ( lio\_listio; aio.h;;;)*
- 1531 **SEE:** Assertion GA\_macro\_args in §2.7.3
- 1532 *Conformance for lio\_listio: PASS, NO\_OPTION*
- 1533 **6.7.4.2 Description**
- 1534 **5** **IF** *PCTS\_lio\_listio* **THEN**
- 1535 **TEST:** A successful call to the *lio\_listio()* function, where the *mode* argument is LIO\_WAIT,
- 1536 waits until all I/O is complete, ignores the *sig* argument, and returns zero.
- 1537 **ELSE** *NO\_OPTION*
- 1538 *Conformance for lio\_listio: PASS, NO\_OPTION*
- 1539 **6** **IF** *PCTS\_lio\_listio* **THEN**
- 1540 **TEST:** A successful call to the *lio\_listio()* function, where the *mode* argument is LIO\_NOWAIT
- 1541 and where the *sig* argument is NULL returns immediately, does not deliver a signal
- 1542 upon completion of all I/O operations, and returns zero.
- 1543 **ELSE** *NO\_OPTION*
- 1544 *Conformance for lio\_listio: PASS, NO\_OPTION*
- 1545 **7** **IF** *PCTS\_lio\_listio* **THEN**
- 1546 **TEST:** A successful call to the *lio\_listio()* function, where the *mode* argument is LIO\_NOWAIT,
- 1547 and where the *sig* argument is not NULL and the *sigev\_signo* member of the *sigevent*
- 1548 structure referenced by *sig* is zero, returns immediately, does not deliver a signal upon
- 1549 completion of all I/O operations, and returns zero.
- 1550 **ELSE** *NO\_OPTION*
- 1551 *Conformance for lio\_listio: PASS, NO\_OPTION*
- 1552 **8** **IF** *PCTS\_lio\_listio* **THEN**
- 1553 **TEST:** A successful call to the *lio\_listio()* function, where the *mode* argument is LIO\_NOWAIT
- 1554 and where the *sig* argument is not NULL and the *sigev\_signo* member of the *sigevent*
- 1555 structure referenced by *sig* is not zero, the signal number indicated by *sigev\_signo* is
- 1556 delivered when all the requests in *list* have completed and the call returns a value of
- 1557 zero immediately.
- 1558 **ELSE** *NO\_OPTION*
- 1559 *Conformance for lio\_listio: PASS, NO\_OPTION*
- 1560 **D\_1** **IF** *PCTS\_lio\_listio* and a PCD.1b documents the following **THEN**
- 1561 **TEST:** A PCD.1b that documents the order in which the I/O requests enumerated by *list* are
- 1562 submitted does so in §6.7.4.2.
- 1563 **ELSE** *NO\_OPTION*
- 1564 *Conformance for lio\_listio: PASS, NO\_OPTION*
- 1565 **9** **IF** *PCTS\_lio\_listio* **THEN**
- 1566 **TEST:** A successful call to the *lio\_listio()* function with NULL elements in the *list* argument
- 1567 ignores the NULL elements and returns to zero.



1568                   **TR:** Test with NULL elements interspersed in the *list*.  
1569                   **ELSE NO\_OPTION**  
1570                   Conformance for *lio\_listio*: PASS, NO\_OPTION

1571   **10**           **IF PCTS\_lio\_listio THEN**  
1572                   **TEST:** A successful call to the *lio\_listio*() function where the  *aio\_lio\_opcode* field of an  
1573                                     *aiocb* structure specifies the LIO\_NOP operation causes the list entry to be ignored.  
1574                   **TR:** Test with multiple file descriptors and buffers in a single call with LIO\_NOP operations.  
1575                   **ELSE NO\_OPTION**  
1576                   Conformance for *lio\_listio*: PASS, NO\_OPTION

1577   **lio\_read\_op**  
1578                   **IF PCTS\_lio\_listio THEN**  
1579                   **TEST:** A successful call to the *lio\_listio*() function where the  *aio\_lio\_opcode* field of an  
1580                                     *aiocb* structure specifies the LIO\_READ operation causes an I/O operation to be  
1581                                    submitted as if by a call to  *aio\_read* with the  *aiocb* structure.  
1582                   **TR:** Test with multiple file descriptors and buffers in a single call. If  
1583                                    POSIX\_PRIORITY\_SCHEDULING, test with multiple scheduling priorities.  
1584                   **ELSE NO\_OPTION**  
1585                   Conformance for *lio\_listio*: PASS, NO\_OPTION

1586   **lio\_write\_op**  
1587                   **IF PCTS\_lio\_listio THEN**  
1588                   **TEST:** A successful call to the *lio\_listio*() function where the  *aio\_lio\_opcode* field of an  
1589                                     *aiocb* structure specifies the LIO\_WRITE operation causes an I/O operation to be  
1590                                    submitted as if by a call to  *aio\_write* with the  *aiocb* equal to the address of the  *aiocb*  
1591                                    structure.  
1592                   **TR:** Test with multiple file descriptors and buffers in a single call. If  
1593                                    POSIX\_PRIORITY\_SCHEDULING, test with multiple scheduling priorities.  
1594                   **ELSE NO\_OPTION**  
1595                   Conformance for *lio\_listio*: PASS, NO\_OPTION

1596   **R\_1 IF PCTS\_lio\_listio THEN**  
1597                   **TEST:** The *list* element further describes the I/O operation to be performed, in a manner  
1598                                    identical to that of the corresponding  *aiocb* structure when used by the  *aio\_read*() and  
1599                                     *aio\_write* () functions.  
1600                   **ELSE NO\_OPTION**  
1601                   **SEE:** The *lio\_read\_op* and *lio\_write\_op* assertions.

1602   **11**           **IF PCTS\_lio\_listio and { \_POSIX\_SYNCHRONIZED\_IO } THEN**  
1603                   **SETUP:** Open a file by calling  *open*() with O\_RSYNC and O\_DSYNC set in the  *oflag* parameter.  
1604                   **TEST:** A read operation initiated by calling *lio\_listio*() either completes by transferring an  
1605                                    image of the data to the requesting process, or if unsuccessful, by diagnosing and  
1606                                    returning an indicator of the error.  
1607                   **TR:** Test for regular files and, if PCTS\_GTI\_DEVICE, terminals.  
1608                   **NOTE:** There is no known portable test method for this assertion.  
1609                   **ELSE NO\_OPTION**  
1610                   **SEE:** Assertion GA\_syncIODataIntegrityRead in §2.2.2.119  
1611                   Conformance for *lio\_listio*: PASS, NO\_TEST, NO\_OPTION

1612   **12**           **IF PCTS\_lio\_listio and { \_POSIX\_SYNCHRONIZED\_IO } THEN**  
1613                   **SETUP:** Open a file by calling  *open*() with O\_RSYNC and O\_SYNC set in the  *oflag* parameter.  
1614                   **TEST:** At the time that the synchronized read operation initiated by calling *lio\_listio*() occurs,  
1615                                    any pending write requests affecting the data to be read are written to the physical  
1616                                    medium containing the file prior to reading the data and the following file attributes  
1617                                    are also written to the physical medium containing the file prior to returning to the  
1618                                    calling process:  
1619                                    1. File mode.

- 1620 2. File serial number.
- 1621 3. ID of device containing this file.
- 1622 4. Number of links.
- 1623 5. User ID of the owner of the file.
- 1624 6. Group ID of the group of the file.
- 1625 7. The file size in bytes.
- 1626 8. Time of last access.
- 1627 9. Time of last data modification.
- 1628 10. Time of last file status change.
- 1629 **TR:** Test for regular files .
- 1630 **NOTE:** There is no known portable test method for this assertion.
- 1631 **ELSE NO\_OPTION**
- 1632 **SEE:** Assertion GA\_syncIOFileIntegrityRead in §2.2.2.120
- 1633 *Conformance for lio\_listio: PASS, NO\_TEST, NO\_OPTION*
- 1634 **14 IF PCTS\_lio\_listio and { \_POSIX\_SYNCHRONIZED\_IO } THEN**
- 1635 **SETUP:** Open a file by calling *open()* with O\_DSYNC set in the *oflag* parameter.
- 1636 **TEST:** A write operation initiated by calling *lio\_listio()* either completes by transferring an
- 1637 image of the data to the physical medium containing the file or, if unsuccessful, by
- 1638 diagnosing and returning an indicator of the error.
- 1639 **TR:** Test with for regular files and, if PCTS\_GTI\_DEVICE, terminals.
- 1640 **NOTE:** There is no known portable test method for this assertion.
- 1641 **ELSE NO\_OPTION**
- 1642 **SEE:** Assertion GA\_syncIODataIntegrityWrite in §2.2.2.119
- 1643 *Conformance for lio\_listio: PASS, NO\_TEST, NO\_OPTION*
- 1644 **15 IF PCTS\_lio\_listio and { \_POSIX\_SYNCHRONIZED\_IO } THEN PCTS\_lio\_listio and**
- 1645 **{ \_POSIX\_SYNCHRONIZED\_IO }**
- 1646 **SETUP:** Open a file by calling *open()* with O\_SYNC set in the *oflag* parameter.
- 1647 **TEST:** At the time that the synchronized write operation initiated by calling *lio\_listio()*
- 1648 occurs, the data are written to the physical medium containing the file and the
- 1649 following file attributes are also written to the physical medium containing the file
- 1650 prior to returning to the calling process.
- 1651 1. File mode.
- 1652 2. File serial number.
- 1653 3. ID of device containing this file.
- 1654 4. Number of links.
- 1655 5. User ID of the owner of the file.
- 1656 6. Group ID of the group of the file.
- 1657 7. The file size in bytes.
- 1658 8. Time of last access.
- 1659 9. Time of last data modification.

1660 10. Time of last file status change.  
 1661 **TR:** Test for regular files.  
 1662 **NOTE:** There is no known portable test method for this assertion.  
 1663 **ELSE NO\_OPTION**  
 1664 **SEE:** Assertion GA\_syncIOFileIntegrityWrite in §2.2.2.120  
 1665 *Conformance for lio\_listio: PASS, NO\_TEST, NO\_OPTION*

1666 **6.7.4.3 Returns**

1667 **R\_2 IF PCTS\_lio\_listio THEN**  
 1668 **TEST:** A call to the *lio\_listio()* function where the *mode* argument has the value LIO\_NOWAIT  
 1669 returns the value zero and queues the I/O operations.  
 1670 **ELSE NO\_OPTION**  
 1671 **SEE:** Assertions in §6.7.4.2.

1672 **R\_3 IF PCTS\_lio\_listio THEN**  
 1673 **TEST:** An unsuccessful call to the *lio\_listio()* function where the *mode* argument has the  
 1674 value LIO\_NOWAIT returns the value -1 and sets *errno* to indicate the error.  
 1675 **ELSE NO\_OPTION**  
 1676 **SEE:** Assertions in §6.7.4.4.

1677 **R\_4 IF PCTS\_lio\_listio THEN**  
 1678 **TEST:** A successful call to the *lio\_listio()* function where the *mode* argument has the value  
 1679 LIO\_WAIT returns the value zero when all the indicated I/O has completed successfully.  
 1680 **ELSE NO\_OPTION**  
 1681 **SEE:** Assertions in §6.7.4.2.

1682 **R\_5 IF PCTS\_lio\_listio THEN**  
 1683 **TEST:** An unsuccessful call to the *lio\_listio()* function where the *mode* argument has the  
 1684 value LIO\_WAIT returns the value -1 and sets *errno* to indicate the error.  
 1685 **ELSE NO\_OPTION**  
 1686 **SEE:** Assertions in §6.7.4.4.

1687 **16 IF PCTS\_lio\_listio THEN**  
 1688 **TEST:** A successful call to the *lio\_listio()* function where some individual requests fail does  
 1689 not prevent completion of any other individual request and returns zero.  
 1690 **TR:** Intersperse individual I/O operations that will fail with those that will succeed.  
 1691 **ELSE NO\_OPTION**  
 1692 *Conformance for lio\_listio: PASS, NO\_OPTION*

1693 **R\_6 IF PCTS\_lio\_listio THEN**  
 1694 **TEST:** The error statuses returned by a call to *lio\_listio()* with the *aio\_lio\_opcode* field of an  
 1695 *aioch* structure equal to LIO\_READ or LIO\_WRITE are identical to those returned as the  
 1696 result of an *aio\_read()* or *aio\_write()* function, respectively.  
 1697 **ELSE NO\_OPTION**  
 1698 **SEE:** Assertions starting with *lio\_read* and *lio\_write* in §6.7.4.4.

1699 **6.7.4.4 Errors**

1700 **17 IF PCTS\_lio\_listio THEN**  
 1701 **SETUP:** Queue enough asynchronous I/O operations before calling *lio\_listio()* to perform the  
 1702 test so that not all I/O requests in the call will have a resource to be queued.  
 1703 **TEST:** A call to the *lio\_listio()* function returns -1 and sets *errno* to [EAGAIN].  
 1704 **TR:** Test with both LIO\_READ and LIO\_WRITE operations.  
 1705 **NOTE:** There is no known portable test method for this assertion.  
 1706 **ELSE NO\_OPTION**  
 1707 *Conformance for lio\_listio: PASS, NO\_TEST, NO\_OPTION*

1708 **18** **IF** *PCTS\_lio\_listio* **THEN**  
 1709     **IF** {AIO\_MAX} ≤ *PCTS\_AIO\_MAX* **THEN**  
 1710         **SETUP:** Queue {AIO\_MAX} or fewer asynchronous I/O operations before calling  
 1711             *lio\_listio()* to perform the test.  
 1712         **TEST:** A call to the *lio\_listio()* function returns -1 and sets *errno* to [EAGAIN].  
 1713         **TR:** Test with both {AIO\_MAX} and fewer queued operations. When testing with less than  
 1714             {AIO\_MAX} operations queued, make sure that some, but not all, of the I/O  
 1715             operations in the call can be queued. Test with both LIO\_READ and LIO\_WRITE  
 1716             operations.  
 1717         **NOTE:** There is no known portable test method for this assertion.  
 1718         **ELSE** *NO\_TEST\_SUPPORT*  
 1719     **ELSE** *NO\_OPTION*  
 1720     *Conformance for lio\_listio: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1721 **19** **IF** *PCTS\_lio\_listio* **THEN**  
 1722     **IF** {AIO\_MAX} ≤ *PCTS\_AIO\_MAX* and *PCTS\_GTI\_DEVICE* **THEN**  
 1723         **SETUP:** Queue {AIO\_MAX} asynchronous I/O operations before calling *lio\_listio()* for the  
 1724             test.  
 1725         **TEST:** A call to the *lio\_listio()* function returns -1 and sets *errno* to [EAGAIN].  
 1726         **TR:** Test for a character special device. Test with both {AIO\_MAX} and fewer queued  
 1727             operations. When testing with less than {AIO\_MAX} operations queued, make sure  
 1728             that some, but not all, of the I/O operations in the call can be queued. Test with both  
 1729             LIO\_READ and LIO\_WRITE operations.  
 1730         **ELSE** *NO\_TEST\_SUPPORT*  
 1731     **ELSE** *NO\_OPTION*  
 1732     *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1733 **20** **IF** *PCTS\_lio\_listio* **THEN**  
 1734     **IF** {AIO\_MAX} > *PCTS\_AIO\_MAX* and *PCTS\_GTI\_DEVICE* **THEN**  
 1735         **SETUP:** Queue *PCTS\_AIO\_MAX - nent* asynchronous I/O operations before calling  
 1736             *lio\_listio()*.  
 1737         **TEST:** A call to the *lio\_listio()* function with *nent* entries returns 0 and does not set  
 1738             *errno* to [EAGAIN].  
 1739         **TR:** Test for a character special device. Test with both LIO\_READ and LIO\_WRITE  
 1740             operations.  
 1741         **ELSE** *NO\_TEST\_SUPPORT*  
 1742     **ELSE** *NO\_OPTION*  
 1743     *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1744 NOTE: Is there anything special about the [EAGAIN] error that depends on *nent*?

1745 **lio\_read\_ebadf1**  
 1746     **IF** *PCTS\_lio\_listio* **THEN**  
 1747         **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
 1748             **TEST:** After a call to the *lio\_listio()* function with an *list[]->aio\_lio\_opcode* equal to  
 1749             LIO\_READ and where the *list[]->aio\_fildes* argument is not a valid file  
 1750             descriptor, the implementation either detects the condition synchronously and  
 1751             then the *lio\_listio()* function returns -1 and sets *errno* to [EIO] or it detects the  
 1752             condition asynchronously and then the return status of the asynchronous  
 1753             operation is set to -1, and the error status of the asynchronous operation is set  
 1754             to [EIO].  
 1755             **ELSE** *NO\_TEST\_SUPPORT*  
 1756         **ELSE** *NO\_OPTION*  
 1757         *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1758 **lio\_read\_ebadf2**  
 1759     **IF** *PCTS\_lio\_listio* **THEN**  
 1760         **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**

1761                   **TEST:**     After a call to the *lio\_listio()* function with an *list[]-> aio\_lio\_opcode* equal to  
 1762                   LIO\_READ and *list[]-> aio\_fildes* argument is not open for reading, the  
 1763                   implementation either detects the condition synchronously and then the  
 1764                   *lio\_listio()* function returns -1 and sets *errno* to [EIO] or it detects the condition  
 1765                   asynchronously and then the return status of the asynchronous operation is set  
 1766                   to -1, and the error status of the asynchronous operation is set to [EIO] .  
 1767                   **ELSE NO\_TEST\_SUPPORT**  
 1768                   **ELSE NO\_OPTION**  
 1769                   *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1770   **lio\_read\_invall**  
 1771                   **IF PCTS\_lio\_listio THEN**  
 1772                    **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**  
 1773                      **TEST:**     After a call to the *lio\_listio()* function with an *list[]-> aio\_lio\_opcode* equal to  
 1774                      LIO\_READ and the file offset value implied by *list[]-> aio\_offset* would be  
 1775                      invalid, *list[]-> aio\_reqprio* is not a valid value, or *list[]-> aio\_nbytes* is an  
 1776                      invalid value, the implementation either detects the condition synchronously and  
 1777                      then the *lio\_listio()* function returns -1 and sets *errno* to [EIO] or it detects the  
 1778                      condition asynchronously and then the return status of the asynchronous  
 1779                      operation is set to [EIO] .  
 1780                      **TR:** Test separately for each of the three conditions above.  
 1781                      **ELSE NO\_TEST\_SUPPORT**  
 1782                      **ELSE NO\_OPTION**  
 1783                      *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1784   **26**           **IF PCTS\_lio\_listio and { \_POSIX\_JOB\_CONTROL } THEN**  
 1785                    **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**  
 1786                      **SETUP:**   Make the testing process part of the background process group and make the  
 1787                      process group of the process orphaned. Call to the *lio\_listio()* function with an  
 1788                      *list[]-> aio\_lio\_opcode* equal to LIO\_READ and so that it successfully queues the  
 1789                      I/O operation to read from its controlling terminal.  
 1790                      **TEST:**     The read operation will fail and the return status of the asynchronous operation  
 1791                      returned by a call to the *aio\_return ()* function is -1 and the error status returned  
 1792                      by a call to the *aio\_error()* function is [EIO].  
 1793                      **ELSE NO\_TEST\_SUPPORT**  
 1794                      **ELSE NO\_OPTION**  
 1795                      *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1796   **27**           **IF PCTS\_lio\_listio and { \_POSIX\_JOB\_CONTROL } THEN**  
 1797                    **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**  
 1798                      **SETUP:**   Make the testing process part of the background process group and have the  
 1799                      process ignore or block the SIGTTIN signal. Call to the *lio\_listio()* function with  
 1800                      an *list[]-> aio\_lio\_opcode* equal to LIO\_READ and so that it successfully queues  
 1801                      the I/O operation to read from its controlling terminal.  
 1802                      **TEST:**     The read operation will fail and the return status of the asynchronous operation  
 1803                      returned by a call to the *aio\_return ()* function is -1 and the error status returned  
 1804                      by a call to the *aio\_error()* function is [EIO].  
 1805                      **ELSE NO\_TEST\_SUPPORT**  
 1806                      **ELSE NO\_OPTION**  
 1807                      *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1808   **R\_7** **IF PCTS\_lio\_listio THEN**  
 1809                    **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**  
 1810                      **TEST:**     After a call to the *lio\_listio()* function with an *list[]-> aio\_lio\_opcode* equal to  
 1811                      LIO\_READ and where the *list[]-> aio\_fildes* argument is not a valid file  
 1812                      descriptor, the implementation either detects the condition synchronously and

1813 the *lio\_listio()* function returns -1 and sets *errno* to [EIO] or it detects the  
 1814 condition asynchronously and the return status of the asynchronous operation is  
 1815 set to -1, and the error status of the asynchronous operation is set to [EIO].

1816 **ELSE NO\_TEST\_SUPPORT**

1817 **ELSE NO\_OPTION**

1818 **SEE:** Assertion *lio\_read\_ebadf1* in §6.7.4.4

1819 **R\_8 IF PCTS\_lio\_listio THEN**

1820 **IF PCTS\_aio\_error THEN**

1821 **TEST:** After a call to the *lio\_listio()* function with an *list[]->aio\_lio\_opcode* equal to  
 1822 LIO\_READ and where the *list[]->aio\_fildes* argument is not open for reading, the  
 1823 implementation either detects the condition synchronously and then the  
 1824 *lio\_listio()* function returns -1 and sets *errno* to [EBADF] or it detects the  
 1825 condition asynchronously and then the return status of the asynchronous  
 1826 operation is set to -1, and the error status of the asynchronous operation is set  
 1827 to [EBADF].

1828 **ELSE NO\_TEST\_SUPPORT**

1829 **ELSE NO\_OPTION**

1830 **SEE:** Assertion *lio\_read\_ebadf2* in §6.7.4.4.

1831 **28 IF PCTS\_lio\_listio and PCTS\_aio\_cancel THEN**

1832 **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**

1833 **SETUP:** Call to the *lio\_listio()* function with a *list[]->aio\_lio\_opcode* equal to LIO\_READ  
 1834 and so that it successfully queues the I/O operation.

1835 **TEST:** A read operation that is canceled after a call to the *aio\_cancel()* function and  
 1836 before the I/O completed results in a return status returned by a call to the  
 1837 *aio\_return()* function that is -1 and the error status returned by a call to the  
 1838 *aio\_error()* function that is [ECANCELED].

1839 **TR:** Test for test for a pipe and a FIFO.

1840 If *PCTS\_GTI\_DEVICE*,, test for a character special file.

1841 **ELSE NO\_TEST\_SUPPORT**

1842 **ELSE NO\_OPTION**

1843 *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1844 **R\_9 IF PCTS\_lio\_listio THEN**

1845 **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**

1846 **SETUP:** After a call to the *lio\_listio()* function with an *list[]->aio\_lio\_opcode* equal to  
 1847 LIO\_READ and where the file offset value implied by *list[]->aio\_offset* would  
 1848 be invalid, the implementation detects the condition asynchronously and then  
 1849 the return status of the asynchronous operation is set to -1, and the error status  
 1850 of the asynchronous operation is set to [EINVAL].

1851 **TEST:** The read operation will fail and the return status of the asynchronous operation  
 1852 returned by a call to the *aio\_return()* function is -1 and the error status returned  
 1853 by a call to the *aio\_error()* function is [EIO].

1854 **ELSE NO\_TEST\_SUPPORT**

1855 **ELSE NO\_OPTION**

1856 **SEE:** Assertion *lio\_read\_einval* in §6.7.4.4.

1857 **lio\_write\_ebadf1**

1858 **IF PCTS\_lio\_listio THEN**

1859 **IF PCTS\_aio\_error and PCTS\_aio\_return THEN**

1860 **TEST:** After a call to the *lio\_listio()* function with an *list[]->aio\_lio\_opcode* equal to  
 1861 LIO\_WRITE and where the *list[]->aio\_fildes* argument is not a valid file  
 1862 descriptor, the implementation either detects the condition synchronously and  
 1863 then the *lio\_listio()* function returns -1 and sets *errno* to [EIO] or it detects the  
 1864 condition asynchronously and then the return status of the asynchronous  
 1865 operation is set to -1, and the error status of the asynchronous operation is set  
 1866 to [EIO].

1867                   **ELSE NO\_TEST\_SUPPORT**  
1868                   **ELSE NO\_OPTION**  
1869                   *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1870   **lio\_write\_ebadf2**  
1871           **IF PCTS\_lio\_listio THEN**  
1872               **IF PCTS\_lio\_error and PCTS\_lio\_return THEN**  
1873                   **TEST:**   After a call to the *lio\_listio()* function with an *list[]->lio\_opcode* equal to  
1874                                   LIO\_WRITE and where the *list[]->lio\_fildes* argument that is not open for  
1875                                   writing, the implementation either detects the condition synchronously and then  
1876                                   the *lio\_listio()* function returns -1 and sets *errno* to [EIO] or it detects the  
1877                                   condition asynchronously and then the return status of the asynchronous  
1878                                   operation is set to -1, and the error status of the asynchronous operation is set  
1879                                   to [EIO].  
1880                   **ELSE NO\_TEST\_SUPPORT**  
1881                   **ELSE NO\_OPTION**  
1882                   *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1883   **lio\_write\_einval1**  
1884           **IF PCTS\_lio\_listio THEN**  
1885               **IF PCTS\_lio\_error and PCTS\_lio\_return THEN**  
1886                   **TEST:**   After a call to the *lio\_listio()* function with an *list[]->lio\_opcode* equal to  
1887                                   LIO\_WRITE and file offset value implied by *list[]->lio\_offset* would be invalid,  
1888                                   the implementation either detects the condition synchronously and then the  
1889                                   *lio\_listio()* function returns -1 and sets *errno* to [EIO] or it detects the condition  
1890                                   asynchronously and then the return status of the asynchronous operation is set  
1891                                   to [EIO].  
1892                   **TR:** Test separately for each of the three conditions above.  
1893                   **ELSE NO\_TEST\_SUPPORT**  
1894                   **ELSE NO\_OPTION**  
1895                   *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1896                                   If *PCTS\_GTI\_DEVICE*, test for a character special file.  
1897                   **ELSE NO\_TEST\_SUPPORT**  
1898                   **ELSE NO\_OPTION**  
1899                   *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1900   **31**           **IF PCTS\_lio\_listio THEN**  
1901               **IF PCTS\_lio\_cancel and PCTS\_lio\_error and PCTS\_lio\_return THEN**  
1902                   **SETUP:** Call to the *lio\_listio()* function with an *list[]->lio\_opcode* equal to  
1903                                   LIO\_WRITE and so that it successfully queues the I/O operation.  
1904                   **TEST:**   After a write operation is canceled by a call to the *lio\_cancel()* function and no  
1905                                   data were transferred, the return status of the asynchronous operation returned  
1906                                   by a call to the *lio\_return()* function is -1 and the error status returned by a call  
1907                                   to the *lio\_error()* function is [EINTR] or [ECANCELED].  
1908                   **TR:** Test for test for a pipe and a FIFO.

1909                                   If *PCTS\_GTI\_DEVICE*, test for a character special file.  
1910                   **ELSE NO\_TEST\_SUPPORT**  
1911                   **ELSE NO\_OPTION**  
1912                   *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1913   **32**           **IF PCTS\_lio\_listio THEN**  
1914               **IF** the implementation supports a maximum file size **THEN**  
1915                   **TEST:**   An attempt to write a file that would exceed an implementation-defined  
1916                                   maximum size by calling the *lio\_listio()* function and where the write operation

1917 is successfully queued, causes the error status for the asynchronous operation to  
 1918 be [EFBIG].  
 1919 **NOTE:** The assertion test would require an unreasonable amount of time or resources  
 1920 on most implementations.  
 1921 **ELSE NO\_TEST\_SUPPORT**  
 1922 **ELSE NO\_OPTION**  
 1923 *Conformance for lio\_listio: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1924 **33** **IF** *PCTS\_lio\_listio* and {\_POSIX\_JOB\_CONTROL} **THEN**  
 1925 **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
 1926 **SETUP:** Make the testing process part of the background process group, set TOSTOP,  
 1927 ignore or block SIGTTOU signals, and make the process group of the process  
 1928 orphaned. Call the *lio\_listio()* function so that it successfully queues the I/O  
 1929 operation to write from its controlling terminal.  
 1930 **TEST:** The write operation will fail and the return status of the asynchronous operation  
 1931 returned by a call to the *aio\_return()* function is -1 and the error status returned  
 1932 by a call to the *aio\_error()* function is [EIO].  
 1933 **ELSE NO\_TEST\_SUPPORT**  
 1934 **ELSE NO\_OPTION**  
 1935 *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1936 **34** **IF** *PCTS\_lio\_listio* **THEN**  
 1937 **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* and *PCTS\_DETECT\_ENOSPC* **THEN**  
 1938 **SETUP:** Fill a device so that there is no more space available on it for data. Call the  
 1939 *lio\_listio()* function so that it successfully queues a write I/O operation.  
 1940 **TEST:** After a call to the *lio\_listio()* function the implementation either detects the  
 1941 condition synchronously and the *lio\_listio()* function returns -1 and sets *errno*  
 1942 to [EIO] or it detects the condition asynchronously and the return status of the  
 1943 asynchronous operation is set to -1, and the error status of the asynchronous  
 1944 operation is set to [EIO].  
 1945 **ELSE NO\_TEST\_SUPPORT**  
 1946 **ELSE NO\_OPTION**  
 1947 *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1948 **35** **IF** *PCTS\_lio\_listio* **THEN**  
 1949 **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
 1950 **SETUP:** Call the *lio\_listio()* function so that it successfully queues a write I/O operation.  
 1951 **TEST:** After a call to the *lio\_listio()* function to write to a pipe or FIFO that is not open  
 1952 for reading by any process, the implementation either detects the condition  
 1953 synchronously and the *lio\_listio()* function returns -1 and sets *errno* to [EIO] or  
 1954 it detects the condition asynchronously and the return status of the asynchronous  
 1955 operation is set to -1, and the error status of the asynchronous operation is set  
 1956 to [EIO].  
 1957 **ELSE NO\_TEST\_SUPPORT**  
 1958 **ELSE NO\_OPTION**  
 1959 *Conformance for lio\_listio: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1960 **R\_10** **IF** *PCTS\_lio\_listio* **THEN**  
 1961 **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
 1962 **TEST:** After a call to the *lio\_listio()* function where the *list[]->aio\_fildes* argument is  
 1963 not a valid file descriptor, the implementation detects the condition  
 1964 asynchronously and then the return status of the asynchronous operation is set  
 1965 to -1, and the error status of the asynchronous operation is set to [EBADF].  
 1966 **ELSE NO\_TEST\_SUPPORT**  
 1967 **ELSE NO\_OPTION**  
 1968 **SEE:** Assertion *lio\_write\_ebadf1* in §6.7.4.4.

1969 **R\_11** **IF** *PCTS\_lio\_listio* **THEN**  
 1970 **IF** *PCTS\_aio\_error* **THEN**



1971                   **TEST:**     After a call to the *lio\_listio()* function where the *list[]->aio\_fildes* argument is  
1972                   not open for writing, the implementation detects the condition synchronously  
1973                   and then the *lio\_listio()* function returns -1 and sets *errno* to [EBADF] or it  
1974                   detects the condition asynchronously and then the return status of the  
1975                   asynchronous operation is set to -1, and the error status of the asynchronous  
1976                   operation is set to [EBADF].  
1977                   **ELSE NO\_TEST\_SUPPORT**  
1978                   **ELSE NO\_OPTION**  
1979                   **SEE:**       Assertion *lio\_write\_ebadf2* in §6.7.4.4.

1980                   **36**       **IF** *PCTS\_lio\_listio* and *PCTS\_aio\_cancel* **THEN**  
1981                   **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
1982                   **SETUP:**    Call the *lio\_listio()* function so that it successfully queues a write I/O operation.  
1983                   **TEST:**       A write function that is canceled after a call to the *aio\_cancel()* function and  
1984                   before the I/O completed results in a return status returned by a call to the  
1985                   *aio\_return()* function that is -1 and the error status returned by a call to the  
1986                   *aio\_error()* function that is [ECANCELED].  
1987                   **TR:**       Test for test for a pipe and a FIFO.

1988                   If *PCTS\_GTI\_DEVICE*, test for a character special file.  
1989                   **ELSE NO\_TEST\_SUPPORT**  
1990                   **ELSE NO\_OPTION**  
1991                   Conformance for *lio\_listio*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1992                   **R-12 IF** *PCTS\_lio\_listio* **THEN**  
1993                   **IF** *PCTS\_aio\_error* and *PCTS\_aio\_return* **THEN**  
1994                   **TEST:**       After a call to the *lio\_listio()* function for a write operation where the file offset  
1995                   value implied by *list[]->aio\_offset* would be invalid, the implementation  
1996                   detects the condition asynchronously and then the return status of the  
1997                   asynchronous operation is set to -1, and the error status of the asynchronous  
1998                   operation is set to [EINVAL].  
1999                   **ELSE NO\_TEST\_SUPPORT**  
2000                   **ELSE NO\_OPTION**  
2001                   **SEE:**       Assertion *lio\_write\_einval1* in §6.7.4.4.

2002                   **37**       **IF** *PCTS\_lio\_listio* **THEN**  
2003                   **TEST:**       A call to the *lio\_listio()* where the *mode* argument is not a proper value returns -1 and  
2004                   sets *errno* to [EINVAL].  
2005                   **ELSE NO\_OPTION**  
2006                   Conformance for *lio\_listio*: PASS, NO\_OPTION

2007                   **38**       **IF** *PCTS\_lio\_listio* **THEN**  
2008                   **TEST:**       A call to the *lio\_listio()* where the value of *nent* is greater than {AIO\_LISTIO\_MAX},  
2009                   returns -1 and sets *errno* to [EINVAL].  
2010                   **ELSE NO\_OPTION**  
2011                   Conformance for *lio\_listio*: PASS, NO\_OPTION

2012                   **39**       **IF** *PCTS\_lio\_listio* **THEN**  
2013                   **SETUP:**    Call to *lio\_listio()* where the *mode* argument is LIO\_WAIT and there are at least two  
2014                   I/O operations and at least one of which will complete before the others and send a  
2015                   signal indicating its completion.  
2016                   **TEST:**       Such an *lio\_listio()* call will receive a signal while waiting for all I/O requests to  
2017                   complete during an operation and return -1 and set *errno* to [EINTR] and outstanding  
2018                   I/O requests are not canceled.  
2019                   **TR:**       Test for a signal also generated by another process.  
2020                   **ELSE NO\_OPTION**  
2021                   Conformance for *lio\_listio*: PASS, NO\_OPTION

2022                   **40**       **IF** not *PCTS\_lio\_listio* **THEN**

2023                   **TEST:**     A call to the *lio\_listio()* function returns -1 and sets *errno* to [ENOSYS].  
 2024                   **ELSE NO\_OPTION**  
 2025                   *Conformance for lio\_listio: PASS, NO\_OPTION*

2026           **6.7.5 Retrieve Error Status of Asynchronous I/O Operation**

2027           Function: *aio\_error()*

2028           **6.7.5.1 Synopsis**

2029           **1**

2030                   *M\_GA\_stdC\_proto\_decl(int; aio\_error; const struct aiocb \* aiocbp; aio.h;;)*  
 2031                   **SEE:**       Assertion GA\_stdC\_proto\_decl in §2.7.3  
 2032                   *Conformance for aio\_error: PASS[1, 2], NO\_OPTION*

2033           **2**

2034                   *M\_GA\_commonC\_int\_result\_decl(aio\_error; aio.h;;)*  
 2035                   **SEE:**       Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 2036                   *Conformance for aio\_error: PASS[1, 2], NO\_OPTION*

2037           **3**

2038                   *M\_GA\_macro\_result\_decl(int; aio\_error; aio.h;;)*  
 2039                   **SEE:**       Assertion GA\_macro\_result\_decl in §1.3.4  
 2040                   *Conformance for aio\_error: PASS, NO\_OPTION*

2041           **4**

2042                   *M\_GA\_macro\_args ( aio\_error; aio.h;;)*  
 2043                   **SEE:**       Assertion GA\_macro\_args in §2.7.3  
 2044                   *Conformance for aio\_error: PASS, NO\_OPTION*

2045           **6.7.5.2 Description**

2046           **5**       **IF PCTS\_aio\_error THEN**  
 2047                   **TEST:**     A call to the *aio\_error()* function returns the error status associated with the *aiocb*  
 2048                                   structure referenced by the *aiocbp* argument.  
 2049                   **ELSE NO\_OPTION**  
 2050                   *Conformance for aio\_error: PASS, NO\_OPTION*

2051           **R\_1 IF PCTS\_aio\_error THEN**  
 2052                   **TEST:**     The error status returned by a call to *aio\_error()* is the *errno* value that would be set  
 2053                                   by the corresponding *read()*, *write()*, or *fsync()* operation.  
 2054                   **ELSE NO\_OPTION**  
 2055                   **SEE:**       Assertions for *aio\_read()* in §6.7.2.4.

2056           **6**       **IF PCTS\_aio\_error THEN**  
 2057                   **TEST:**     A call to the *aio\_error()* function where the *aiocb* structure referenced by the *aiocbp*  
 2058                                   argument refers to an operation that has not yet completed returns the value  
 2059                                   [EINPROGRESS].  
 2060                   **ELSE NO\_OPTION**  
 2061                   *Conformance for aio\_error: PASS, NO\_OPTION*

2062           **6.7.5.3 Returns**

2063           **7**       **IF PCTS\_aio\_error THEN**  
 2064                   **TEST:**     A call to the *aio\_error()* function where the *aiocb* structure referenced by the *aiocbp*  
 2065                                   argument refers to an asynchronous I/O operation that has completed successfully,  
 2066                                   returns 0.

2067           **ELSE NO\_OPTION**  
 2068           *Conformance for aio\_error: PASS, NO\_OPTION*

2069    **R-2**    **TEST:**    The error status returned by a call to *aio\_error()* for an asynchronous I/O operation that has  
 2070                                    completed unsuccessfully is the *errno* value that would be set by the corresponding *read()*,  
 2071                                    *write()*, or *fsync()* operation.  
 2072           **ELSE NO\_OPTION**  
 2073           **SEE:**        Assertions for *aio\_read()* in §6.7.2.4.

2074    **8**        **IF PCTS\_aio\_error THEN**  
 2075                    **TEST:**    A call to the *aio\_error()* function where the *aiocb* structure referenced by the *aiocbp*  
 2076                                    argument refers to an asynchronous I/O operation has not yet completed returns  
 2077                                    [EINPROGRESS].  
 2078           **ELSE NO\_OPTION**  
 2079           *Conformance for aio\_error: PASS, NO\_OPTION*

2080    **6.7.5.4 Errors**

2081    **9**        **IF not PCTS\_aio\_error THEN**  
 2082                    **TEST:**    A call to the *aio\_error()* returns -1 and sets *errno* to [ENOSYS].  
 2083           **ELSE NO\_OPTION**  
 2084           *Conformance for aio\_error: PASS, NO\_OPTION*

2085    **10**       **IF PCTS\_aio\_error THEN**  
 2086                    **IF PCTS\_DETECT\_AIO\_ERROR\_AIOCBP THEN**  
 2087                                    **TEST:**    A call to the *aio\_error()* where the *aiocbp* argument refers to an asynchronous  
 2088                                    operation whose return status has previously been retrieved returns -1 and sets  
 2089                                    *errno* to [EINVAL].  
 2090                    **ELSE NO\_TEST\_SUPPORT**  
 2091           **ELSE NO\_OPTION**  
 2092           *Conformance for aio\_error: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2093    **11**       **IF PCTS\_aio\_error THEN**  
 2094                    **IF PCTS\_DETECT\_AIO\_ERROR\_AIOCBP THEN**  
 2095                                    **TEST:**    A call to the *aio\_error()* function where the *aiocbp* argument refers to an invalid  
 2096                                    *aiocb* structure returns -1 and sets *errno* to [EINVAL].  
 2097                    **ELSE NO\_TEST\_SUPPORT**  
 2098           **ELSE NO\_OPTION**  
 2099           *Conformance for aio\_error: PASS, NO\_OPTION*

## 2100    **6.7.6 Retrieve Return Status of Asynchronous I/O Operation**

2101    Function: *aio\_return()*

### 2102    **6.7.6.1 Synopsis**

2103    **1**  
 2104            *M\_GA\_stdC\_proto\_decl(int; aio\_return; struct aiocb \* aiocbp; aio.h;;;)*  
 2105            **SEE:**        Assertion GA\_stdC\_proto\_decl in §2.7.3  
 2106            *Conformance for aio\_return: PASS[1, 2], NO\_OPTION*

2107    **2**  
 2108            *M\_GA\_commonC\_int\_result\_decl(ssize\_t; aio\_return; aio.h;;;)*  
 2109            **SEE:**        Assertion GA\_commonC\_result\_decl in §2.7.3  
 2110            *Conformance for aio\_return: PASS[1, 2], NO\_OPTION*

2111    **3**

2112 *M\_GA\_macro\_result\_decl(ssize\_t; aio\_return; aio.h;;)*  
 2113 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 2114 *Conformance for aio\_return: PASS, NO\_OPTION*

2115 **4**

2116 *M\_GA\_macro\_args ( aio\_return; aio.h;;)*  
 2117 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 2118 *Conformance for aio\_return: PASS, NO\_OPTION*

### 2119 **6.7.6.2 Description**

#### 2120 **return\_status**

2121 **IF PCTS\_aio\_return THEN**  
 2122 **TEST:** A call to the *aio\_return()* function returns the return status associated with the *aiocb*  
 2123 structure referenced by the *aiocbp* argument.  
 2124 **ELSE NO\_OPTION**  
 2125 *Conformance for aio\_return: PASS, NO\_OPTION*

2126 **R\_1 IF PCTS\_aio\_return THEN**  
 2127 **TEST:** The return status for an asynchronous I/O operation is the value that would be returned  
 2128 by the corresponding *read()*, *write()*, or *fsync()* function call.  
 2129 **ELSE NO\_OPTION**  
 2130 **SEE:** Assertions for *aio\_read()* in §6.7.2.4.

2131 **D\_1 IF PCTS\_aio\_return and a PCD.1b documents the following THEN**  
 2132 **TEST:** A PCD.1b that documents the return status for an operation returned by a call to  
 2133 *aio\_return()* when the error status is equal to [EINPROGRESS] does so in §6.7.6.2.  
 2134 **ELSE NO\_OPTION**  
 2135 *Conformance for aio\_return: PASS, NO\_OPTION*

2136 **R\_2 IF PCTS\_aio\_return THEN**  
 2137 **SETUP:** Initiate an asynchronous I/O operation and let it complete. Retrieve the return status  
 2138 by calling *aio\_return()*.  
 2139 **TEST:** A call to the *aio\_return()* or the *aio\_error()* function using the same *aiocb* structure  
 2140 as used in a previously successful call to *aio\_return()* returns -1 and sets *errno* to  
 2141 [EINVAL].  
 2142 **TR:** Test for calling both *aio\_return()* and *aio\_error()*.  
 2143 **ELSE NO\_OPTION**  
 2144 **SEE:** Assertion *EINVAL* in §6.7.6.4.

2145 **5 IF PCTS\_aio\_return THEN**  
 2146 **SETUP:** Initiate and complete an asynchronous I/O operation. Call the *aio\_return()* function to  
 2147 retrieve its return status. Submit the same *aiocb* structure referred to by *aiocbp* in the  
 2148 *aio\_return()* call to another asynchronous operation.  
 2149 **TEST:** A call to the *aio\_return()* function can reuse an *aiocb* structure used in a previously  
 2150 successful *aio\_return()* call after it has been used to initiate another asynchronous I/O  
 2151 operation.  
 2152 **ELSE NO\_OPTION**  
 2153 *Conformance for aio\_return: PASS, NO\_OPTION*

### 2154 **6.7.6.3 Returns**

2155 **R\_3 IF PCTS\_aio\_return THEN**  
 2156 **SETUP:** After an asynchronous I/O operation has completed, the return status, as described for  
 2157 *read()*, *write()*, and *fsync()*, is returned.  
 2158 **ELSE NO\_OPTION**  
 2159 **SEE:** Assertion *return\_status* in §6.7.6.4.

2160 **D\_2** **IF** a PCD.1b documents the following **THEN**  
 2161       **TEST:** A PCD.1b that documents defines the results of a call to the *aio\_return()* function  
 2162               when the asynchronous I/O operation has not yet completed does so in §6.7.6.4.  
 2163       **ELSE NO\_OPTION**  
 2164       Conformance for *aio\_return*: *PASS, NO\_OPTION*

#### 2165 6.7.6.4 Errors

##### 2166 **EINVAL**

2167       **IF** *PCTS\_aio\_return* **THEN**  
 2168           **SETUP:** Initiate an asynchronous I/O operation and let it complete. Retrieve the return status  
 2169                       by calling *aio\_return()*.  
 2170           **TEST:** A call to the *aio\_return()* or the *aio\_error()* function using the same *aioCB* structure  
 2171                       as used in a previously successful call to *aio\_return()* returns -1 and sets *errno* to  
 2172                       [EINVAL].  
 2173           **TR:** Test for calling *aio\_return()*. If *PCTS\_DETECT\_AIO\_ERROR\_AIOCBP* then test for *aio\_error()*.  
 2174       **ELSE NO\_OPTION**  
 2175       Conformance for *aio\_return*: *PASS, NO\_OPTION*

2176 **6**       **IF** *PCTS\_aio\_return* **THEN**  
 2177           **TEST:** A call to the *aio\_return()* function where the *aioCBP* argument refers to an invalid  
 2178                       *aioCB* structure returns -1 and sets *errno* to [EINVAL].  
 2179       **ELSE NO\_OPTION**  
 2180       Conformance for *aio\_return*: *PASS, NO\_OPTION*

2181 **7**       **IF** not *PCTS\_aio\_return* **THEN**  
 2182           **TEST:** A call to the *aio\_return()* function returns -1 and sets *errno* to [ENOSYS].  
 2183       **ELSE NO\_OPTION**  
 2184       Conformance for *aio\_return*: *PASS, NO\_OPTION*

#### 2185 6.7.7 Cancel Asynchronous I/O Request

2186 Function: *aio\_cancel()*

##### 2187 6.7.7.1 Synopsis

2188 **1**  
 2189       *M\_GA\_stdC\_proto\_decl(int; aio\_cancel; int fildes, struct aioCB \* aioCBP; aio.h;;)*  
 2190       **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 2191       Conformance for *aio\_cancel*: *PASS[1, 2], NO\_OPTION*

2192 **2**  
 2193       *M\_GA\_commonC\_int\_result\_decl(aio\_cancel; aio.h;;)*  
 2194       **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 2195       Conformance for *aio\_cancel*: *PASS[1, 2], NO\_OPTION*

2196 **3**  
 2197       *M\_GA\_macro\_result\_decl(int; aio\_cancel; aio.h;;)*  
 2198       **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 2199       Conformance for *aio\_cancel*: *PASS, NO\_OPTION*

2200 **4**  
 2201       *M\_GA\_macro\_args ( aio\_cancel; aio.h;;)*  
 2202       **SEE:** Assertion GA\_macro\_args in §2.7.3  
 2203       Conformance for *aio\_cancel*: *PASS, NO\_OPTION*

2204 **6.7.7.2 Description**

2205 **5** **IF** *PCTS\_aid\_cancel* **THEN**  
2206 **IF** *PCTS\_AIO\_CANCELABLE\_OPS* and ( *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lid\_listio* ) **THEN**  
2207 **SETUP:** Queue asynchronous I/O requests for one file descriptor that will not complete  
2208 before calling the *aid\_cancel*() function.  
2209 **TEST:** A call to the *aid\_cancel*() function will cancel the asynchronous I/O request  
2210 currently outstanding against file descriptor *filde*s specified by the *aiocbp*  
2211 argument and return the value AIO\_CANCELED.  
2212 **TR:** Test by queuing a single request. Then test by queueing multiple requests each of  
2213 which uses a different *aiocbp* against a single file descriptor. Then test by queueing  
2214 multiple requests against multiple file descriptors.  
2215 **ELSE NO\_TEST\_SUPPORT**  
2216 **ELSE NO\_OPTION**  
2217 *Conformance for aid\_cancel:* PASS, NO\_TEST\_SUPPORT, NO\_OPTION

2218 **6** **IF** *PCTS\_aid\_cancel* **THEN**  
2219 **IF** *PCTS\_AIO\_CANCELABLE\_OPS* and ( *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lid\_listio* ) **THEN**  
2220 **SETUP:** Queue multiple asynchronous I/O requests for multiple file descriptors that will  
2221 not complete signal notification before calling the *aid\_cancel*() function.  
2222 **TEST:** A call to the *aid\_cancel*() function where the *aiocbp* argument is NULL cancels  
2223 the outstanding cancellable asynchronous I/O requests against the file descriptor  
2224 *filde*s and returns the value AIO\_CANCELED, and normal signal delivery occurs  
2225 for the canceled operations..  
2226 **ELSE NO\_TEST\_SUPPORT**  
2227 **ELSE NO\_OPTION**  
2228 *Conformance for aid\_cancel:* PASS, NO\_TEST\_SUPPORT, NO\_OPTION

2229 **7** **IF** *PCTS\_aid\_cancel* **THEN**  
2230 **IF** *PCTS\_AIO\_CANCELABLE\_OPS* and ( *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lid\_listio* ) **THEN**  
2231 **TEST:** Any asynchronous I/O requests that cannot be canceled by a call to the  
2232 *aid\_cancel*() function complete following the normal asynchronous completion  
2233 process.  
2234 **NOTE:** There is no known portable test method for this assertion.  
2235 **ELSE NO\_TEST\_SUPPORT**  
2236 **ELSE NO\_OPTION**  
2237 *Conformance for aid\_cancel:* PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION

2238 **8** **IF** *PCTS\_aid\_cancel* **THEN**  
2239 **IF** *PCTS\_AIO\_CANCELABLE\_OPS* and ( *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lid\_listio* ) **THEN**  
2240 **TEST:** After a call to the *aid\_cancel*() function, for requested operations that are  
2241 successfully canceled, the associated error status is set to [ECANCELED] and the  
2242 return status is -1.  
2243 **TR:** Test for the *aiocbp* pointing to a valid *aiocb* and for it being NULL.  
2244 **ELSE NO\_TEST\_SUPPORT**  
2245 **ELSE NO\_OPTION**  
2246 *Conformance for aid\_cancel:* PASS, NO\_TEST\_SUPPORT, NO\_OPTION

2247 **9** **IF** *PCTS\_aid\_cancel* **THEN**  
2248 **IF** *PCTS\_AIO\_CANCELABLE\_OPS* and ( *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lid\_listio* ) **THEN**  
2249 **TEST:** After a call to the *aid\_cancel*() function, for requested operations that are not  
2250 successfully canceled, the *aiocbp* is not modified by *aid\_cancel*().  
2251 **ELSE NO\_TEST\_SUPPORT**  
2252 **ELSE NO\_OPTION**  
2253 *Conformance for aid\_cancel:* PASS, NO\_TEST\_SUPPORT, NO\_OPTION

2254 **D\_1** **IF** *PCTS\_aid\_cancel* and a PCD.1b documents the following **THEN**

2255                   **TEST:**     A PCD.1b that documents the results of calling *aio\_cancel()* where the *aioctx* is not  
 2256                                    **NULL** and the *fildev* argument does not have the same value as the file descriptor with  
 2257                                    which the asynchronous operation was initiated does so in §6.7.7.2.  
 2258                   **ELSE NO\_OPTION**  
 2259                   *Conformance for aio\_cancel: PASS, NO\_OPTION*

2260   **D\_2 IF PCTS\_aio\_cancel THEN**  
 2261                   **TEST:**     The PCD.1b documents which asynchronous i/o operations are cancellable by calling  
 2262                                    the *aio\_cancel()* function in §6.7.7.2.  
 2263                   **ELSE NO\_OPTION**  
 2264                   *Conformance for aio\_cancel: PASS, NO\_OPTION*

### 2265   **6.7.7.3 Returns**

2266   **R\_2 IF PCTS\_aio\_cancel THEN**  
 2267                   **TEST:**     The *aio\_cancel()* function returns the value **AIO\_CANCELED** to the calling process if  
 2268                                    the requested operation(s) were canceled.  
 2269                   **ELSE NO\_OPTION**  
 2270                   **SEE:**        Assertions in §6.7.7.2.

2271   **10 IF PCTS\_aio\_cancel THEN**  
 2272                   **IF PCTS\_AIO\_CANCELABLE\_OPS and ( PCTS\_aio\_read or PCTS\_aio\_write or PCTS\_lio\_listio) THEN**  
 2273                   **TEST:**     After a call to the *aio\_cancel()* function, the value **AIO\_NOTCANCELED** is  
 2274                                    returned when at least one of the requested operation(s) cannot be canceled  
 2275                                    because it is in progress.  
 2276                   **NOTE:**     There is no known portable test method for assertion.  
 2277                   **ELSE NO\_TEST\_SUPPORT**  
 2278                   **ELSE NO\_OPTION**  
 2279                   *Conformance for aio\_cancel: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

2280   **11 IF PCTS\_aio\_cancel THEN**  
 2281                   **IF PCTS\_aio\_read or PCTS\_aio\_write or PCTS\_lio\_listio THEN**  
 2282                   **TEST:**     The value **AIO\_ALLDONE** is returned for a call to the *aio\_cancel()* function when  
 2283                                    all of the operations have already completed before they could be canceled.  
 2284                   **ELSE NO\_TEST\_SUPPORT**  
 2285                   **ELSE NO\_OPTION**  
 2286                   *Conformance for aio\_cancel: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2287   **R\_3 IF PCTS\_aio\_cancel THEN**  
 2288                   **TEST:**     When an error occurs in a call to the *aio\_cancel()* function it returns -1 and sets *errno*  
 2289                                    to indicate the error.  
 2290                   **ELSE NO\_OPTION**  
 2291                   **SEE:**        Assertions in §6.7.7.4.

### 2292   **6.7.7.4 Errors**

2293   **12 IF PCTS\_aio\_cancel THEN**  
 2294                   **IF PCTS\_AIO\_CANCELABLE\_OPS and ( PCTS\_aio\_read or PCTS\_aio\_write or PCTS\_lio\_listio) THEN**  
 2295                   **TEST:**     A call to the *aio\_cancel()* function where the *fildev* argument is not a valid file  
 2296                                    descriptor returns -1 and sets *errno* to [EBADF].  
 2297                   **ELSE NO\_TEST\_SUPPORT**  
 2298                   **ELSE NO\_OPTION**  
 2299                   *Conformance for aio\_cancel: PASS, NO\_OPTION*

2300   **13 IF not PCTS\_aio\_cancel THEN**  
 2301                   **TEST:**     A call to the *aio\_cancel()* function returns -1 and sets *errno* to [ENOSYS].  
 2302                   **ELSE NO\_OPTION**

2303 *Conformance for aio\_cancel: PASS, NO\_OPTION*

## 2304 **6.7.8 Wait for Asynchronous I/O Request**

2305 Function: *aio\_suspend()*

### 2306 **6.7.8.1 Synopsis**

2307 **1**

*M\_GA\_stdC\_proto\_decl(int; aio\_suspend; const struct aiocb \* const list[], int nent, const struct timespec \*timeout; aio.h;;;)*

2310 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3

2311 *Conformance for aio\_suspend: PASS[1, 2], NO\_OPTION*

2312 **2**

*M\_GA\_commonC\_int\_result\_decl(aio\_suspend; aio.h;;;)*

2314 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3

2315 *Conformance for aio\_suspend: PASS[1, 2], NO\_OPTION*

2316 **3**

*M\_GA\_macro\_result\_decl(int; aio\_suspend; aio.h;;;)*

2318 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4

2319 *Conformance for aio\_suspend: PASS, NO\_OPTION*

2320 **4**

*M\_GA\_macro\_args ( aio\_suspend; aio.h;;;)*

2322 **SEE:** Assertion GA\_macro\_args in §2.7.3

2323 *Conformance for aio\_suspend: PASS, NO\_OPTION*

### 2324 **6.7.8.2 Description**

#### 2325 **completion**

2326 **IF** *PCTS\_aio\_suspend* **THEN**

2327 **IF** *PCTS\_aio\_read* or *PCTS\_aio\_write* or *PCTS\_lio\_listio* **THEN**

2328 **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.

2330 **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 zero.

2333 **ELSE** *NO\_TEST\_SUPPORT*

2334 **ELSE** *NO\_OPTION*

2335 *Conformance for aio\_suspend: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

#### 2336 **interrupt**

2337 **IF** *PCTS\_aio\_suspend* **THEN**

2338 **IF** *PCTS\_aio\_read* or *PCTS\_aio\_write* or *PCTS\_lio\_listio* **THEN**

2339 **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.

2341 **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].

2344 **TR:** Test for a signal coming from the completion of an asynchronous I/O operation and for a signal coming from another process.

2346 **ELSE** *NO\_TEST\_SUPPORT*

2347 **ELSE** *NO\_OPTION*

2348 *Conformance for aio\_suspend: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*



2349 **timeout**

2350 **IF** *PCTS\_aid\_suspend* **THEN**

2351 **IF** *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lio\_listio* **THEN**

2352 **SETUP:** Queue asynchronous I/O operations that will not complete pass the list of the

2353 operations to the *aid\_suspend()* function in the test.

2354 **TEST:** A call to the *aid\_suspend()* function with the *timeout* argument not equal to

2355 **NULL** suspends the calling process until the time interval specified by *timeout*

2356 has passed and returns -1 and sets *errno* to [EAGAIN].

2357 **ELSE NO\_TEST\_SUPPORT**

2358 **ELSE NO\_OPTION**

2359 *Conformance for aid\_suspend: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2360 **already\_completed**

2361 **IF** *PCTS\_aid\_suspend* **THEN**

2362 **IF** *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lio\_listio* **THEN**

2363 **SETUP:** Queue asynchronous I/O operations so that at least one of them will complete

2364 before *aid\_suspend()* is called.

2365 **TEST:** A call to the *aid\_suspend()* function where any of the *aiocb* structures in the *list*

2366 argument correspond to completed asynchronous I/O operations (i.e., the error

2367 status for the operation is not equal to [EINPROGRESS]) at the time of the call

2368 returns without suspending the calling process and returns zero.

2369 **ELSE NO\_TEST\_SUPPORT**

2370 **ELSE NO\_OPTION**

2371 *Conformance for aid\_suspend: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2372 **5**

2373 **IF** *PCTS\_aid\_suspend* **THEN**

2374 **IF** *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lio\_listio* **THEN**

2375 **TEST:** A call to the *aid\_suspend()* function ignores any **NULL** pointers in the *list*

2376 argument.

2377 **ELSE NO\_TEST\_SUPPORT**

2378 **ELSE NO\_OPTION**

2379 *Conformance for aid\_suspend: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2379 **D\_1** **IF** a PCD.1b documents the following **THEN**

2380 **TEST:** A PCD.1b that documents the effect of a call to the *aid\_suspend()* function where the

2381 *list* argument contains pointers that refer to *aiocb* structures that have not been used

2382 in submitting asynchronous I/O does so in §6.7.8.2.

2383 **ELSE NO\_OPTION**

2384 *Conformance for aid\_suspend: PASS, NO\_OPTION*

2385 **R\_1**

2386 **IF** *PCTS\_aid\_suspend* **THEN**

2387 **IF** *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lio\_listio* **THEN**

2388 **TEST:** After a call to the *aid\_suspend()* function and after the time interval indicated

2389 in the *timespec* structure pointed to by *timeout* passes before any of the I/O

2390 operations referenced by *list* are completed, the *aid\_suspend()* returns with an

2391 error.

2392 **ELSE NO\_TEST\_SUPPORT**

2393 **ELSE NO\_OPTION**

2394 **SEE:** Assertion timeout in §6.7.8.2.

2394 **6.7.8.3 Returns**

2395 **R\_2** **IF** *PCTS\_aid\_suspend* **THEN**

2396 **IF** *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lio\_listio* **THEN**

2397 **TEST:** A call to the *aid\_suspend()* function returns after one or more asynchronous I/O

2398 operations have completed and returns to zero.

2399 **ELSE NO\_TEST\_SUPPORT**

2400 **ELSE NO\_OPTION**

2401 **SEE:** Assertions completion and *already\_completed* in §6.7.8.2.

2402 **R\_3 IF PCTS\_aid\_suspend THEN**  
 2403 **IF PCTS\_aid\_read or PCTS\_aid\_write or PCTS\_lid\_listio THEN**  
 2404 **TEST:** A call to the *aid\_suspend()* function returns a value of -1 and sets *errno* to  
 2405 indicate the error when an error condition is detected.  
 2406 **ELSE NO\_TEST\_SUPPORT**  
 2407 **ELSE NO\_OPTION**  
 2408 **SEE:** Assertions timeout and interrupt in §6.7.8.2 and no\_support in §6.7.8.4.

#### 2409 6.7.8.4 Errors

2410 **R\_4 IF PCTS\_aid\_suspend THEN**  
 2411 **IF PCTS\_aid\_read or PCTS\_aid\_write or PCTS\_lid\_listio THEN**  
 2412 **TEST:** A call to the *aid\_suspend()* where no asynchronous I/O indicated in the list  
 2413 referenced by *list* completed in the time interval indicated by *timeout* returns -1  
 2414 and sets *errno* to [EAGAIN].  
 2415 **ELSE NO\_TEST\_SUPPORT**  
 2416 **ELSE NO\_OPTION**  
 2417 **SEE:** Assertion timeout in §6.7.8.2.

2418 **6 TEST:** A call to *aid\_suspend()* where a signal interrupts the *aid\_suspend()* function returns -1 and  
 2419 sets *errno* to [EINTR].  
 2420 **SEE:** Assertion interrupt in §6.7.8.2.  
 2421 *Conformance for aid\_suspend: PASS*

#### 2422 no\_support

2423 **IF not PCTS\_aid\_suspend THEN**  
 2424 **TEST:** A call to the *aid\_suspend()* function returns -1 and sets *errno* to [ENOSYS].  
 2425 **ELSE NO\_OPTION**  
 2426 *Conformance for aid\_suspend: PASS, NO\_OPTION*

#### 2427 6.7.9 Asynchronous File Synchronization

2428 Function: *aid\_fsync()*

##### 2429 6.7.9.1 Synopsis

2430 **1**  
 2431 *M\_GA\_stdC\_proto\_decl(int; aid\_fsync; int op, struct aiocb \* aiocbp; aio.h;;)*  
 2432 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 2433 *Conformance for aid\_fsync: PASS[1, 2], NO\_OPTION*

2434 **2**  
 2435 *M\_GA\_commonC\_int\_result\_decl(aid\_fsync; aio.h;;)*  
 2436 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 2437 *Conformance for aid\_fsync: PASS[1, 2], NO\_OPTION*

2438 **3**  
 2439 *M\_GA\_macro\_result\_decl(int; aid\_fsync; aio.h;;)*  
 2440 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 2441 *Conformance for aid\_fsync: PASS, NO\_OPTION*

2442 **4**  
 2443 *M\_GA\_macro\_args ( aid\_fsync; aio.h;;)*  
 2444 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 2445 *Conformance for aid\_fsync: PASS, NO\_OPTION*

##### 2446 6.7.9.2 Description

- 2447 **5** **IF** *PCTS\_aio\_fsync* **THEN**  
 2448 **IF** *PCTS\_aio\_read* or *PCTS\_lio\_listio* **THEN**  
 2449 **SETUP:** Queue asynchronous read operations using as many of *PCTS\_aio\_read* and  
 2450 *PCTS\_lio\_listio* as are supported by the implementation against a file for which  
 2451 I/O data synchronization has not been set.  
 2452 **TEST:** A call to the *aio\_fsync()* function with the *op* argument equal to O\_DSYNC  
 2453 asynchronously forces all read operations associated with the file indicated by  
 2454 the file descriptor *aio\_fildes* member of the *aiocb* structure referenced by the  
 2455 *aiocbp* argument and queued at the time of the call to *aio\_fsync()* to the  
 2456 synchronized I/O data completion state and returns zero when the  
 2457 synchronization request has been initiated or queued to the file or device. That  
 2458 is, the read operation either completes by transferring an image of the data to  
 2459 the requesting process or, if unsuccessful, by diagnosing and returning an  
 2460 indicator of the error.  
 2461 **NOTE:** There is no known portable test method for this assertion.  
 2462 **ELSE NO\_TEST\_SUPPORT**  
 2463 **ELSE NO\_OPTION**  
 2464 **SEE:** Assertion GA\_syncIODataIntegrityRead in §2.2.2.119  
 2465 *Conformance for aio\_fsync: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 2466 **6** **IF** *PCTS\_aio\_fsync* **THEN**  
 2467 **IF** *PCTS\_aio\_read* or *PCTS\_lio\_listio* **THEN**  
 2468 **SETUP:** Queue asynchronous read operations using as many of *PCTS\_aio\_read* and  
 2469 *PCTS\_lio\_listio* as are supported by the implementation against a file for which  
 2470 I/O data synchronization has not been set.  
 2471 **TEST:** A call to the *aio\_fsync()* function with the *op* argument equal to O\_DSYNC  
 2472 asynchronously forces all read operations associated with the file indicated by  
 2473 the file descriptor *aio\_fildes* member of the *aiocb* structure referenced by the  
 2474 *aiocbp* argument and queued at the time of the call to *aio\_fsync()* to the  
 2475 synchronized I/O data completion state and returns zero when the  
 2476 synchronization request has been initiated or queued to the file or device. That  
 2477 is, at the time that the synchronized read operation initiated by calling  
 2478 *aio\_fsync()* any pending write requests affecting the data to be read are written  
 2479 to the physical medium containing the file prior to reading the data.  
 2480 **NOTE:** There is no known portable test method for this assertion.  
 2481 **ELSE NO\_TEST\_SUPPORT**  
 2482 **ELSE NO\_OPTION**  
 2483 **SEE:** Assertion GA\_syncIODataIntegrityWbeforeR in §2.2.2.119  
 2484 *Conformance for aio\_fsync: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*
- 2485 **7** **IF** *PCTS\_aio\_fsync* **THEN**  
 2486 **IF** *PCTS\_aio\_write* or *PCTS\_lio\_listio* **THEN**  
 2487 **SETUP:** Queue asynchronous write operations using as many of *PCTS\_aio\_write* and  
 2488 *PCTS\_lio\_listio* as are supported by the implementation against a file for which  
 2489 I/O data synchronization has not been set.  
 2490 **TEST:** A call to the *aio\_fsync()* function with the *op* argument equal to O\_DSYNC  
 2491 asynchronously forces all write operations associated with the file indicated by  
 2492 the file descriptor *aio\_fildes* member of the *aiocb* structure referenced by the  
 2493 *aiocbp* argument and queued at the time of the call to *aio\_fsync()* to the  
 2494 synchronized I/O data completion state and returns zero when the  
 2495 synchronization request has been initiated or queued to the file or device. That  
 2496 is, a write operation initiated by calling *aio\_fsync()* either completes by  
 2497 transferring an image of the data to the physical medium containing the file or,  
 2498 if unsuccessful, by diagnosing and returning and indicator of the error.  
 2499 **TR:** Test for regular files and, if PCTS\_GTI\_DEVICE, terminals.  
 2500 **NOTE:** There is no known portable test method for this assertion.  
 2501 **ELSE NO\_TEST\_SUPPORT**  
 2502 **ELSE NO\_OPTION**  
 2503 **SEE:** Assertion GA\_syncIODataIntegrityWrite in §2.2.2.119

2504 *Conformance for aio\_fsync: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2505 **8** **IF** *PCTS\_aio\_fsync* **THEN**

2506 **IF** *PCTS\_aio\_read* or *PCTS\_lio\_listio* **THEN**

2507 **SETUP:** Queue asynchronous read operations using as many of *PCTS\_aio\_read* and

2508 *PCTS\_lio\_listio* as are supported by the implementation against a file for which

2509 I/O data synchronization has not been set.

2510 **TEST:** A call to the *aio\_fsync()* function with the *op* argument equal to *O\_SYNC*

2511 asynchronously forces all read operations associated with the file indicated by

2512 the file descriptor *aio\_fildes* member of the *aiocb* structure referenced by the

2513 *aiocbp* argument and queued at the time of the call to *aio\_fsync()* to the

2514 synchronized I/O data completion state and returns zero when the

2515 synchronization request has been initiated or queued to the file or device. That

2516 is: At the time that the synchronized read operation initiated by calling

2517 *aio\_fsync()* occurs, any pending write requests affecting the data to be read are

2518 written to the physical medium containing the file prior to reading the data and

2519 the following file attributes are also written to the physical medium containing

2520 the file prior to returning to the calling process:

2521 1. File mode.

2522 2. File serial number.

2523 3. ID of device containing this file.

2524 4. Number of links.

2525 5. User ID of the owner of the file.

2526 6. Group ID of the group of the file.

2527 7. The file size in bytes.

2528 8. Time of last access.

2529 9. Time of last data modification.

2530 10. Time of last file status change.

2531 **NOTE:** There is no known portable test method for this assertion.

2532 **ELSE** *NO\_TEST\_SUPPORT*

2533 **ELSE** *NO\_OPTION*

2534 **SEE:** Assertion *GA\_syncIOFileIntegrityRead* in §2.2.2.120

2535 *Conformance for aio\_fsync: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

2536 **9** **IF** *PCTS\_aio\_fsync* **THEN**

2537 **IF** *PCTS\_aio\_write* or *PCTS\_lio\_listio* **THEN**

2538 **SETUP:** Queue asynchronous write operations using as many of *PCTS\_aio\_write* and

2539 *PCTS\_lio\_listio* as are supported by the implementation against a file for which

2540 I/O data synchronization has not been set.

2541 **TEST:** A call to the *aio\_fsync()* function with the *op* argument equal to *O\_SYNC*

2542 asynchronously forces all write operations associated with the file indicated by

2543 the file descriptor *aio\_fildes* member of the *aiocb* structure referenced by the

2544 *aiocbp* argument and queued at the time of the call to *aio\_fsync()* to the

2545 synchronized I/O file completion state and returns zero when the synchronization

2546 request has been initiated or queued to the file or device. That is: At the time

2547 that the synchronized write operation initiated by calling *aio\_fsync()* occurs, the

2548 data are written to the physical medium containing the file and the following file

2549 attributes are also written to the physical medium containing the file prior to

2550 returning to the calling process:

- 2551 1. File mode.
- 2552 2. File serial number.
- 2553 3. ID of device containing this file.
- 2554 4. Number of links.
- 2555 5. User ID of the owner of the file.
- 2556 6. Group ID of the group of the file.
- 2557 7. The file size in bytes.
- 2558 8. Time of last access.
- 2559 9. Time of last data modification.
- 2560 10. Time of last file status change.

2561 **NOTE:** There is no known portable test method for this assertion.

2562 **ELSE NO\_TEST\_SUPPORT**

2563 **ELSE NO\_OPTION**

2564 **SEE:** Assertion GA\_syncIOFileIntegrityWrite in §2.2.2.120

2565 *Conformance for aio\_fsync: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

2566 **10 IF PCTS\_aio\_fsync THEN**

2567 **IF PCTS\_aio\_read or PCTS\_write or PCTS\_lio\_listio THEN**

2568 **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 *aio\_cb* structure referenced by the *aio\_cbp* argument and queued at the time of the call will be [EINPROGRESS], and the *aio\_fsync()* call returns zero.

2570 **TR:** Test for as many of the *PCTS\_aio\_read*, *PCTS\_aio\_write*, and *PCTS\_lio\_listio* as are supported by the implementation.

2572 **ELSE NO\_TEST\_SUPPORT**

2573 **ELSE NO\_OPTION**

2574 *Conformance for aio\_fsync: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2575 **11 IF PCTS\_aio\_fsync THEN**

2576 **IF PCTS\_aio\_read or PCTS\_write or PCTS\_lio\_listio THEN**

2577 **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.

2579 **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.

2581 **ELSE NO\_TEST\_SUPPORT**

2582 **ELSE NO\_OPTION**

2583 *Conformance for aio\_fsync: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2584 **12 IF PCTS\_aio\_fsync THEN**

2585 **IF PCTS\_aio\_read or PCTS\_write or PCTS\_lio\_listio THEN**

2586 **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.

2587 **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 zero.

2597                   **TR:** Test for as many of the *PCTS\_aid\_read*, *PCTS\_aid\_write*, and *PCTS\_lio\_listio* as are  
 2598                   supported by the implementation. Test for the *op* argument being *O\_DSYNC* and  
 2599                   *O\_SYNC*.  
 2600                   **ELSE NO\_TEST\_SUPPORT**  
 2601                   **ELSE NO\_OPTION**  
 2602                   Conformance for *aid\_fsync*: *PASS*, *NO\_TEST\_SUPPORT*, *NO\_OPTION*

2603   **13**       **IF PCTS\_aid\_fsync THEN**  
 2604               **IF PCTS\_aid\_read or PCTS\_aid\_write or PCTS\_lio\_listio THEN**  
 2605               **SETUP:** Queue asynchronous operations, including some write operations using as  
 2606               many of *PCTS\_aid\_read*, *PCTS\_aid\_write*, and *PCTS\_lio\_listio* as are supported  
 2607               by the implementation against a file for which I/O synchronized has not been  
 2608               set.  
 2609               **TEST:** During a call to the *aid\_fsync()* function all members of the structure referenced  
 2610               by *aiocbp* are ignored except for *aid\_fildes* and *aid\_sigevent*.  
 2611               **TR:** Test for as many of the *PCTS\_aid\_read*, *PCTS\_aid\_write*, and *PCTS\_lio\_listio* as are  
 2612               supported by the implementation. Test for the *op* argument being *O\_DSYNC* and  
 2613               *O\_SYNC*.  
 2614               **ELSE NO\_TEST\_SUPPORT**  
 2615               **ELSE NO\_OPTION**  
 2616               Conformance for *aid\_fsync*: *PASS*, *NO\_TEST\_SUPPORT*, *NO\_OPTION*

2617   **D\_1** **IF PCTS\_aid\_fsync** and a PCD.1b documents the following **THEN**  
 2618               **TEST:** A PCD.1b that documents the behavior of an implementation when the control block  
 2619               referenced by *aiocbp* in a call to the *aid\_fsync()* function becomes an illegal address  
 2620               prior to asynchronous I/O completion does so in §6.7.9.2.  
 2621               **ELSE NO\_OPTION**  
 2622               Conformance for *aid\_fsync*: *PASS*, *NO\_OPTION*

2623   NOTE: The following statement in POSIX.1b {3} does not make sense since there is no file to synchronize:

2624   If *aiocbp* is *NULL*, then no status is returned in *aiocbp*, and no signal is generated upon completion of the operation.

### 2625   **6.7.9.3 Returns**

2626   **R\_1** **IF PCTS\_aid\_fsync THEN**  
 2627               **TEST:** The *aid\_fsync()* function returns the value 0 to the calling process when the I/O  
 2628               operation is successfully queued.  
 2629               **ELSE NO\_OPTION**  
 2630               **SEE:** Assertions in §6.7.9.2.

2631   **R\_2** **IF PCTS\_aid\_fsync THEN**  
 2632               **TEST:** The *aid\_fsync()* function returns the value -1 and sets *errno* to indicate the error.  
 2633               **ELSE NO\_OPTION**  
 2634               **SEE:** Assertions in §6.7.9.4.

### 2635   **6.7.9.4 Errors**

2636   **14**       **IF PCTS\_aid\_fsync THEN**  
 2637               **IF {AIO\_MAX} ≤ PCTS\_AIO\_MAX and ( PCTS\_aid\_read or PCTS\_aid\_write or PCTS\_lio\_listio ) THEN**  
 2638               **SETUP:** Queue {AIO\_MAX} asynchronous I/O operations that will not complete for a file  
 2639               that does not have its synchronized I/O attributes set.  
 2640               **TEST:** A call to the *aid\_fsync()* function when the requested asynchronous operation  
 2641               was not queued due to temporary resource limitations returns -1 and sets *errno*  
 2642               to [EAGAIN].

2643                   **TR:** Test for as many of the *PCTS\_aid\_read*, *PCTS\_aid\_write*, and *PCTS\_lid\_listio* as are  
2644                   supported by the implementation. Test for the *op* argument being O\_DSYNC and  
2645                   O\_SYNC.  
2646                   **ELSE NO\_TEST\_SUPPORT**  
2647                   **ELSE NO\_OPTION**  
2648                   *Conformance for aio\_fsync: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2649   **15**           **IF** *PCTS\_aid\_fsync* **THEN**  
2650                   **IF**  $\{AIO\_MAX\} \leq PCTS\_AIO\_MAX$  and ( *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lid\_listio*) **THEN**  
2651                   **SETUP:** Queue *PCTS\_AIO\_MAX* asynchronous I/O operations that will not complete for a  
2652                   file that does not have its synchronized I/O attributes set.  
2653                   **TEST:** A call to the *aio\_fsync()* returns zero and does not set *errno* to [EAGAIN].  
2654                   **TR:** Test for as many of the *PCTS\_aid\_read*, *PCTS\_aid\_write*, and *PCTS\_lid\_listio* as are  
2655                   supported by the implementation. Test for the *op* argument being O\_DSYNC and  
2656                   O\_SYNC.  
2657                   **ELSE NO\_TEST\_SUPPORT**  
2658                   **ELSE NO\_OPTION**  
2659                   *Conformance for aio\_fsync: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2660   **16**           **IF** *PCTS\_aid\_fsync* **THEN**  
2661                   **IF** *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lid\_listio* **THEN**  
2662                   **TEST:** A call to the *aio\_fsync()* where the *aio\_fildes* member of the *aiocb* structure  
2663                   referenced by the *aiocbp* argument is not a valid file descriptor open for writing  
2664                   returns -1 and sets *errno* to [EBADF].  
2665                   **TR:** Test for as many of the *PCTS\_aid\_read*, *PCTS\_aid\_write*, and *PCTS\_lid\_listio* as are  
2666                   supported by the implementation. Test for the *op* argument being O\_DSYNC and  
2667                   O\_SYNC.  
2668                   **ELSE NO\_TEST\_SUPPORT**  
2669                   **ELSE NO\_OPTION**  
2670                   *Conformance for aio\_fsync: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2671   **17**           **IF** *PCTS\_aid\_fsync* **THEN**  
2672                   **IF** *PCTS\_NO\_SYNCH\_IO\_FILE* and ( *PCTS\_aid\_read* or *PCTS\_aid\_write* or *PCTS\_lid\_listio*) **THEN**  
2673                   **SETUP:** Queue asynchronous I/O operations for a file for which the implementation  
2674                   does not support synchronized I/O.  
2675                   **TEST:** A call to the *aio\_fsync()* function for a file for which the implementation does  
2676                   not support synchronized I/O returns -1 and sets *errno* to [EINVAL].  
2677                   **TR:** Test for as many of the *PCTS\_aid\_read*, *PCTS\_aid\_write*, and *PCTS\_lid\_listio* as are  
2678                   supported by the implementation. Test for the *op* argument being O\_DSYNC and  
2679                   O\_SYNC.  
2680                   **ELSE NO\_TEST\_SUPPORT**  
2681                   **ELSE NO\_OPTION**  
2682                   *Conformance for aio\_fsync: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2683   **18**           **IF** *PCTS\_aid\_fsync* **THEN**  
2684                   **TEST:** A call to the *aio\_fsync()* function with a value of *op* other than O\_DSYNC or O\_SYNC  
2685                   returns -1 and sets *errno* to [EINVAL].  
2686                   **ELSE NO\_OPTION**  
2687                   *Conformance for aio\_fsync: PASS, NO\_OPTION*

2688   **19**           **IF** not *PCTS\_aid\_fsync* **THEN**  
2689                   **TEST:** A call to the *aio\_fsync()* function returns -1 and sets *errno* to [ENOSYS].  
2690                   **ELSE NO\_OPTION**  
2691                   *Conformance for aio\_fsync: PASS, NO\_OPTION*

This page is intentionally blank.



## Section 7: Device- and Class- Specific Functions

180 There are no POSIX.1b {3} assertions in Section 7.

## Section 8: Language-Specific Services for the C Programming Language

180 There are no POSIX.1b {3} assertions in Section 8 except for subclause 8.2.2.2.

### 181 8.2.2 Open a Stream on a File Descriptor

182 Function: *fdopen()*

#### 183 8.2.2.1 Synopsis

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

#### 185 8.2.2.2 Description

186 **D\_1** **IF** a PCD.1b documents the following **THEN**

187         **TEST:**     A PCD.1b that documents the result of the *fdopen()* function when *files* refers to a  
188                     shared memory object does so in §8.2.2.3.

189         **ELSE** *NO\_OPTION*

190             *Conformance for fdopen: PASS, NO\_OPTION*

#### 191 8.2.2.3 Returns

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

#### 193 8.2.2.4 Errors

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

## Section 9: System Databases

180      There are no POSIX.1b {3} assertions in Section 9.

## Section 10: Data Interchange Format

180 There are no POSIX.1b {3} assertions in Section 10.

## Section 11: Synchronization

### 180 11.1 Semaphore Characteristics

181 1 **SETUP:** Include <semaphore.h>.

182 **TEST:** The type *sem\_t* is defined.

183 *Conformance for sem.hdr: PASS*

184 *M\_GA\_semOpenMaxFD(PCTS\_SEM\_FILE\_DESCRIPTOR; function; PCTS\_function) =*

185 **IF** *PCTS\_SEM\_FILE\_DESCRIPTOR* **THEN**

186 **IF** (*{OPEN\_MAX}* <= *PCTS\_OPEN\_MAX*) and *PCTS\_function* **THEN**

187 **TEST:** A process calling *function()* can simultaneously open a combination of files and  
188 semaphores totaling at least *{OPEN\_MAX}*.

189 **TR:** Test for opening *{OPEN\_MAX}* semaphores.

190 Test for opening a semaphore after opening *{OPEN\_MAX}* -1 files.

191 Test for opening a file after opening *{OPEN\_MAX}* -1 semaphores.

192 **ELSE** *NO\_TEST\_SUPPORT*

193 **ELSE** *NO\_OPTION*

194 **GA\_semOpenMaxFD**

195 **FOR:** *sem\_init()* and *sem\_open()*

196 *M\_GA\_semOpenMaxFD(PCTS\_SEM\_FILE\_DESCRIPTOR; function; PCTS\_function) =*

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

201 *Conformance for sem\_hdr: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

202 *M\_GA\_semPCTSOpenMaxFD(PCTS\_SEM\_FILE\_DESCRIPTOR; function; PCTS\_function) =*

203 **IF** *PCTS\_SEM\_FILE\_DESCRIPTOR* **THEN**

204 **IF** (*{OPEN\_MAX}* <= *PCTS\_OPEN\_MAX*) and *PCTS\_function* **THEN**

205 **TEST:** A process calling *function()* can simultaneously open a combination of files and  
206 semaphores totaling at least *PCTS\_OPEN\_MAX*.

207 **TR:** Test for opening *PCTS\_OPEN\_MAX*-1 files.

208 Test for opening a file, after opening *{OPEN\_MAX}* -1 files.

209 Test for opening a file after opening *PCTS\_OPEN\_MAX* -1 semaphores.

210 **ELSE** *NO\_TEST\_SUPPORT*

211 **ELSE** *NO\_OPTION*

212 **GasemPCTSOpenMaxFD**

213 **FOR:** *sem\_init()* and *sem\_open()*

214 *M\_GA\_semPCTSOpenMaxFD(PCTS\_SEM\_FILE\_DESCRIPTOR; function; PCTS\_function)*

215 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The assertion is  
216 to be read by substituting *function()* with the current function specified in the FOR clause.

217                   The name of the function also is to be substituted for each occurrence in the construct  
 218                   *PCTS\_function*.  
 219                   Conformance for *sem\_hdr*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

## 220    **11.2    Semaphore Functions**

### 221    **11.2.1    Initialize an Unnamed Semaphore**

222    Function: *sem\_init()*

#### 223    **11.2.1.1 Synopsis**

224    **1**  
 225            *M\_GA\_stdC\_proto\_decl(int; sem\_init; sem\_t \*sem, int pshared, unsigned int value; semaphore.h;;)*  
 226    **SEE:**       Assertion *GA\_stdC\_proto\_decl* in §2.7.3  
 227            Conformance for *sem\_init*: *PASS[1, 2], NO\_OPTION*

228    **2**  
 229            *M\_GA\_commonC\_int\_result\_decl(sem\_init; semaphore.h;;)*  
 230    **SEE:**       Assertion *GA\_commonC\_int\_result\_decl* in §2.7.3  
 231            Conformance for *sem\_init*: *PASS[1, 2], NO\_OPTION*

232    **3**  
 233            *M\_GA\_macro\_result\_decl(int; sem\_init; semaphore.h;;)*  
 234    **SEE:**       Assertion *GA\_macro\_result\_decl* in §1.3.4  
 235            Conformance for *sem\_init*: *PASS, NO\_OPTION*

236    **4**  
 237            *M\_GA\_macro\_args ( sem\_init; semaphore.h;;)*  
 238    **SEE:**       Assertion *GA\_macro\_args* in §2.7.3  
 239            Conformance for *sem\_init*: *PASS, NO\_OPTION*

#### 240    **11.2.1.2 Description**

241    *M\_GA\_semOpenMaxFD(PCTS\_SEM\_INIT\_FD; sem\_init; PCTS\_sem\_init)*  
 242    **SEE:**       Assertion *GA\_semOpenMaxFD* in §11.1  
 243            Conformance for *sem\_init*: *PASS/OpenMaxSems, PCTSopenMaxSems]*

244    *M\_GA\_semPCTSOpenMaxFD(PCTS\_SEM\_INIT\_FD; sem\_init; PCTS\_sem\_init)*  
 245    **SEE:**       Assertion *GA\_semPCTSOpenMaxFD* in §11.1  
 246            Conformance for *sem\_init*: *PASS/OpenMaxSems, PCTSopenMaxSems]*

247    **sem\_init**  
 248            **IF** *PCTS\_sem\_init* **THEN**  
 249                **IF:** *PCTS\_GAP\_sem\_init* **THEN**  
 250                    **TEST:**    A call *sem\_init (sem, pshared, val)* initializes the unnamed semaphore *sem* to  
 251                                the value of *val*.  
 252                    **ELSE** *NO\_TEST\_SUPPORT*  
 253                    **ELSE** *NO\_OPTION*  
 254            Conformance for *sem\_init*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

255    **5**    **FOR:** *sem\_wait(), sem\_trywait(), sem\_post(), and sem\_destroy ()*  
 256            **IF** *PCTS\_sem\_init* **THEN**  
 257                **IF:** *PCTS\_GAP\_sem\_init* and *PCTS\_function* **THEN**  
 258                    **TEST:**    The interface *function()* returns successfully when using a semaphore created  
 259                                by *sem\_init()*.  
 260                    **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
 261                                assertion is to be read by substituting *function()* with the current function

262 specified in the FOR clause. The name of the function also is to be substituted  
 263 for each occurrence in the construct *PCTS\_function*.  
 264 **ELSE NO\_TEST\_SUPPORT**  
 265 **ELSE NO\_OPTION**  
 266 *Conformance for sem\_init: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

267 **6 IF PCTS\_sem\_init THEN**  
 268 **IF: PCTS\_GAP\_sem\_init THEN**  
 269 **TEST:** A semaphore created by *sem\_init()* can be used until it is destroyed.  
 270 **TR:** Create a semaphore with a positive value, decrement to zero, then increment and re-  
 271 decrement to zero.  
 272 **ELSE NO\_TEST\_SUPPORT**  
 273 **ELSE NO\_OPTION**  
 274 *Conformance for sem\_init: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

275 **7 FOR: sem\_wait(), sem\_trywait(), sem\_post(), and sem\_destroy()**  
 276 **IF PCTS\_sem\_init THEN**  
 277 **IF: PCTS\_GAP\_sem\_init and PCTS\_function THEN**  
 278 **SETUP:** Create a semaphore by calling *sem\_init(sem, pshared, val)*, with a non-zero  
 279 value of *pshared*.  
 280 **TEST:** Any process that can access the semaphore *sem* can use it for performing  
 281 *function()*.  
 282 **TR:** Test with values {INT\_MAX} and {INT\_MIN} for *pshared*.  
 283  
 284 Perform *function()* on the semaphore from the same process that created it and  
 285 from a child process for each of these values of *pshared*.  
 286 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
 287 assertion is to be read by substituting *function()* with the current function  
 288 specified in the FOR clause. The name of the function also is to be substituted  
 289 for each occurrence in the construct *PCTS\_function*.  
 290 **ELSE NO\_TEST\_SUPPORT**  
 291 **ELSE NO\_OPTION**  
 292 *Conformance for sem\_init: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

293 **D\_1 IF PCTS\_sem\_init and a PCD.1b documents the following THEN**  
 294 **TEST:** A PCD.1b that documents the result of referring to copies of *sem* in calls to  
 295 *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_destroy()* does so in §11.2.1.2.  
 296 **ELSE NO\_OPTION**  
 297 *Conformance for sem\_init: PASS, NO\_OPTION*

298 **D\_2 IF PCTS\_sem\_init and a PCD.1b documents the following THEN**  
 299 **TEST:** A PCD.1b that documents the result of a *pshared* argument of zero does so in  
 300 §11.2.1.2.  
 301 **ELSE NO\_OPTION**  
 302 *Conformance for sem\_init: PASS, NO\_OPTION*

303 **D\_3 IF PCTS\_sem\_init and a PCD.1b documents the following THEN**  
 304 **TEST:** A PCD.1b that documents whether or not it supports the *sem\_init()* function does so  
 305 in §11.2.1.2.  
 306 **ELSE NO\_OPTION**  
 307 *Conformance for sem\_init: PASS, NO\_OPTION*

308 **11.2.1.3 Returns**

309 **R\_1 IF PCTS\_sem\_init THEN**  
 310 **IF PCTS\_GAP\_sem\_init THEN**  
 311 **TEST:** When a call to *sem\_init()* completes successfully, the semaphore *sem* is  
 312 initialized.  
 313 **ELSE NO\_TEST\_SUPPORT**

```

314         ELSE NO_OPTION
315         SEE:      Assertion sem_init in §11.2.1.2

316     R_2 IF PCTS_sem_init THEN
317         IF PCTS_GAP_sem_init THEN
318             TEST:   When a call to sem_init() completes unsuccessfully, the interface returns a
319                    value of -1, sets errno to indicate the error, and does not initialize the
320                    semaphore.
321             ELSE NO_TEST_SUPPORT
322         ELSE NO_OPTION
323         SEE:      All assertions in §11.2.1.4

324     11.2.1.4 Errors

325     8     IF PCTS_sem_init THEN
326         IF PCTS_GAP_sem_init and {SEM_VALUE_MAX}< {UINT_MAX} THEN
327             TEST:   A call to sem_init(sem, pshared, val) when val exceeds {SEM_VALUE_MAX}
328                    returns a value of -1 and sets errno to [EINVAL].
329             ELSE NO_TEST_SUPPORT
330         ELSE NO_OPTION
331         Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

332     9     IF PCTS_sem_init THEN
333         IF PCTS_GAP_sem_init THEN
334             TEST:   When a resource required to initialize the semaphore has been exhausted, a call
335                    to sem_init() returns a value of -1 and sets errno to [ENOSPC].
336             NOTE:   The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough
337                    to write a portable test.
338             ELSE NO_TEST_SUPPORT
339         ELSE NO_OPTION
340         Conformance for sem_init: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

341     10    IF PCTS_sem_init THEN
342         IF PCTS_GAP_sem_init and PCTS_SEM_IS_FD and ( {OPEN_MAX} < PCTS_OPEN_MAX) THEN
343             SETUP:  Open {OPEN_MAX} files. Try to open a semaphore.
344             TEST:   When a resource required to initialize the semaphore has been exhausted, a call
345                    to sem_init() returns a value of -1 and sets errno to [ENOSPC].
346             ELSE NO_TEST_SUPPORT
347         ELSE NO_OPTION
348         Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

349     11    IF PCTS_sem_init THEN
350         IF PCTS_GAP_sem_init and PCTS_SEM_IS_FD and ( {OPEN_MAX} >= PCTS_OPEN_MAX) THEN
351             SETUP:  Open PCTS_OPEN_MAX files. Try to open a semaphore.
352             TEST:   When a resource required to initialize the semaphore has been exhausted, a call
353                    to sem_init() returns a value of -1 and sets errno to [ENOSPC].
354             ELSE NO_TEST_SUPPORT
355         ELSE NO_OPTION
356         Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

357     12    IF PCTS_sem_init THEN
358         IF PCTS_GAP_sem_init and {SEM_NSEMS_MAX}< PCTS_SEM_NSEMS_MAX THEN
359             TEST:   A call to sem_init(), after {SEM_NSEMS_MAX} semaphores have been created,
360                    returns a value of -1 and sets errno to [ENOSPC].
361             ELSE NO_TEST_SUPPORT
362         ELSE NO_OPTION
363         Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

```



364 **13** **IF** *PCTS\_sem\_init* **THEN**  
 365 **IF** *PCTS\_GAP\_sem\_init* and {*SEN\_NSEMS\_MAX*}>= *PCTS\_SEM\_NSEMS\_MAX* **THEN**  
 366 **TEST:** A call to *sem\_init()*, after {*PCTS\_SEM\_NSEMS\_MAX*} semaphores have been  
 367 created, returns a value of -1 and sets *errno* to [ENOSPC].  
 368 **ELSE** *NO\_TEST\_SUPPORT*  
 369 **ELSE** *NO\_OPTION*  
 370 *Conformance for sem\_init: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

371 **14** **IF** not *PCTS\_sem\_init* **THEN**  
 372 **IF** *PCTS\_GAP\_sem\_init* **THEN**  
 373 **TEST:** A call to *sem\_init()* returns a value of -1 and sets *errno* to [ENOSYS].  
 374 **ELSE** *NO\_TEST\_SUPPORT*  
 375 **ELSE** *NO\_OPTION*  
 376 *Conformance for sem\_init: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

377 **15** **IF** *PCTS\_sem\_init* **THEN**  
 378 **IF** *PCTS\_RAP\_sem\_init* **THEN**  
 379 **TEST:** A call to *sem\_init()*, when the process lacks the appropriate privileges to  
 380 initialize a semaphore, returns a value of -1 and sets *errno* to [EPERM].  
 381 **ELSE** *NO\_TEST\_SUPPORT*  
 382 **ELSE** *NO\_OPTION*  
 383 *Conformance for sem\_init: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

### 384 11.2.2 Destroy an Unnamed Semaphore

385 Function: *sem\_destroy()*

#### 386 11.2.2.1 Synopsis

387 **1**  
 388 *M\_GA\_stdC\_proto\_decl(int; sem\_destroy; sem\_t \*sem, semaphore.h;;;)*  
 389 **SEE:** Assertion *GA\_stdC\_proto\_decl* in §2.7.3  
 390 *Conformance for sem\_destroy: PASS[1, 2], NO\_OPTION*

391 **2**  
 392 *M\_GA\_commonC\_int\_result\_decl(sem\_destroy; semaphore.h;;;)*  
 393 **SEE:** Assertion *GA\_commonC\_int\_result\_decl* in §2.7.3  
 394 *Conformance for sem\_destroy: PASS[1, 2], NO\_OPTION*

395 **3**  
 396 *M\_GA\_macro\_result\_decl(int; sem\_destroy; semaphore.h;;;)*  
 397 **SEE:** Assertion *GA\_macro\_result\_decl* in §1.3.4  
 398 *Conformance for sem\_destroy: PASS, NO\_OPTION*  
 399

400 **4**  
 401 *M\_GA\_macro\_args ( sem\_destroy; semaphore.h;;;)*  
 402 **SEE:** Assertion *GA\_macro\_args* in §2.7.3  
 403 *Conformance for sem\_destroy: PASS, NO\_OPTION*

#### 404 11.2.2.2 Description

##### 405 **sem\_destroy**

406 **IF** *PCTS\_sem\_destroy* **THEN**  
 407 **TEST:** A call *sem\_destroy(sem)* destroys the unnamed semaphore *sem* and returns 0.  
 408 **ELSE** *NO\_OPTION*  
 409 *Conformance for sem\_destroy: PASS, NO\_OPTION*

410 **D\_1** **IF** *PCTS\_sem\_destroy* and a PCD.1b documents the following **THEN**

411                   **TEST:**     A PCD.1b that documents the effect of calling *sem\_destroy()* with a named semaphore  
 412                                    does so in §11.2.2.2.  
 413                   **ELSE NO\_OPTION**  
 414                   *Conformance for sem\_destroy: PASS, NO\_OPTION*

415 **D\_2 IF** *PCTS\_sem\_destroy* and a PCD.1b documents the following **THEN**  
 416                   **TEST:**     A PCD.1b that documents the effect of using the semaphore *sem* after it is destroyed  
 417                                    by a call to *sem\_destroy()* does so in §11.2.2.2.  
 418                   **ELSE NO\_OPTION**  
 419                   *Conformance for sem\_destroy: PASS, NO\_OPTION*

420 **D\_3 IF** *PCTS\_sem\_destroy* and a PCD.1b documents the following **THEN**  
 421                   **TEST:**     A PCD.1b that documents the effect of destroying a semaphore upon which other  
 422                                    processes are currently blocked does so in §11.2.2.2.  
 423                   **ELSE NO\_OPTION**  
 424                   *Conformance for sem\_destroy: PASS, NO\_OPTION*

425 **D\_4 IF** *PCTS\_sem\_destroy* and a PCD.1b documents the following **THEN**  
 426                   **TEST:**     A PCD.1b that documents whether or not it supports the *sem\_destroy()* function does  
 427                                    so in §11.2.2.2.  
 428                   **ELSE NO\_OPTION**  
 429                   *Conformance for sem\_destroy: PASS, NO\_OPTION*

### 430 11.2.2.3 Returns

431 **R\_1 IF** *PCTS\_sem\_destroy* **THEN**  
 432                   **TEST:**     When a call to *sem\_destroy()* completes successfully, the interface returns a value of  
 433                                    0.  
 434                   **ELSE NO\_OPTION**  
 435                   **SEE:**     Assertion *sem\_destroy* in §11.2.2.2

436 **R\_2 IF** *PCTS\_sem\_destroy* **THEN**  
 437                   **TEST:**     When a call to *sem\_destroy()* completes unsuccessfully, interface returns a value of  
 438                                    -1 and sets *errno* to indicate the error.  
 439                   **ELSE NO\_OPTION**  
 440                   **SEE:**     All assertions in §11.2.2.4

### 441 11.2.2.4 Errors

442 **5 IF** *PCTS\_sem\_destroy* **THEN**  
 443                   **TEST:**     The call *sem\_destroy(sem)*, when the argument *sem* is not a valid semaphore, returns  
 444                                    a value of -1 and sets *errno* to [EINVAL].  
 445                   **NOTE:**     A subroutine is recommended that either returns an invalid semaphore or indicates  
 446                                    that there is no way to generate an invalid semaphore on the system.  
 447                   **ELSE NO\_OPTION**  
 448                   *Conformance for sem\_destroy: PASS, NO\_OPTION*

449 **6 IF** not *PCTS\_sem\_destroy* **THEN**  
 450                   **TEST:**     A call to *sem\_destroy()* returns a value of -1 and sets *errno* to [ENOSYS].  
 451                   **ELSE NO\_OPTION**  
 452                   *Conformance for sem\_destroy: PASS, NO\_OPTION*

453 **7 IF** *PCTS\_sem\_destroy* and *PCTS\_SEM\_EBUSY* **THEN**  
 454                   **TEST:**     A call to *sem\_destroy(sem)*, when there are currently processes blocked on the  
 455                                    semaphore, returns a value of -1 and sets *errno* to [EBUSY].  
 456                   **ELSE NO\_OPTION**  
 457                   *Conformance for sem\_destroy: PASS, NO\_OPTION*

458 **11.2.3 Initialize/Open a Named Semaphore**459 Function: *sem\_open()*460 **11.2.3.1 Synopsis**461 **1**

462 *M\_GA\_stdC\_proto\_decl(sem\_t\*; sem\_open; const char \*name, int oflag, ...; semaphore.h;;)*  
 463 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 464 *Conformance for sem\_open: PASS[1, 2], NO\_OPTION*

465 **2**

466 *M\_GA\_commonC\_result\_decl(sem\_t\*; sem\_open; semaphore.h;;)*  
 467 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 468 *Conformance for sem\_open: PASS[1, 2], NO\_OPTION*

469 **3**

470 *M\_GA\_macro\_result\_decl(sem\_t\*; sem\_open; semaphore.h;;)*  
 471 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 472 *Conformance for sem\_open: PASS, NO\_OPTION*

473 **4**

474 *M\_GA\_macro\_args ( sem\_open; semaphore.h;;)*  
 475 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 476 *Conformance for sem\_open: PASS, NO\_OPTION*

477 **11.2.3.2 Description**478 *M\_GA\_semOpenMaxFD(PCTS\_SEM\_OPEN\_FD; sem\_open; PCTS\_sem\_open)*

479 **SEE:** Assertion GA\_semOpenMaxFD in §11.1  
 480 *Conformance for sem\_open: PASS/OpenMaxSems, PCTSopenMaxSems]*

481 *M\_GA\_semPCTSOpenMaxFD(PCTS\_SEM\_OPEN\_FD; sem\_open; PCTS\_sem\_open)*

482 **SEE:** Assertion GA\_semPCTSOpenMaxFD in §11.1  
 483 *Conformance for sem\_open: PASS/OpenMaxSems, PCTSopenMaxSems]*

484 **sem\_open**485 **FOR:** *sem\_wait()*, *sem\_trywait()*, *sem\_post()*, and *sem\_close()*486 **IF** *PCTS\_sem\_open* **THEN**487 **IF** *PCTS\_function* **THEN**

488 **TEST:** A call *sem\_open(name, oflag, ...)* establishes a connection between a named  
 489 semaphore, *name*, and the calling process and returns the address of the named  
 490 semaphore that can be used in subsequent calls to *function()*.

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

495 **ELSE** *NO\_TEST\_SUPPORT*496 **ELSE** *NO\_OPTION*497 *Conformance for sem\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*498 **IF** *PCTS\_sem\_open* **THEN**499 **SETUP:** Include the header *<semaphore.h>*.500 **TEST:** The constants *O\_CREAT* and *O\_EXCL* are defined and are bitwise distinct.501 **ELSE** *NO\_OPTION*502 *Conformance for sem\_open: PASS, NO\_OPTION*503 **FOR:** *sem\_close()*, *\_exit()*, and *execl()*, *execv()*, *execle()*, *execve()*, *execlp()*, and *execvp()*

504           **IF** *PCTS\_sem\_open* **THEN**  
505               **SETUP:** Create a semaphore using *sem\_open()*.  
506               **TEST:** The semaphore remains usable by the calling process until it is closed by a successful  
507               call to *function()*.  
508               **TR:** Test using *sem\_wait()*, *sem\_trywait()*, and *sem\_post()*  
509               **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
510               assertion is to be read by substituting *function()* with the current function specified  
511               in the FOR clause. The name of the function also is to be substituted for each  
512               occurrence in the construct *PCTS\_function*.  
513           **ELSE NO\_OPTION**  
514           Conformance for *sem\_open*: *PASS, NO\_OPTION*

515    7       **IF** *PCTS\_sem\_open* **THEN**  
516               **SETUP:** Open a semaphore by calling *sem\_open(name, oflag, ... )* where the flag bit *O\_CREAT*  
517               is set in *oflag* and the named semaphore *name* does not already exist.  
518               **TEST:** The named semaphore is created.  
519           **ELSE NO\_OPTION**  
520           Conformance for *sem\_open*: *PASS, NO\_OPTION*

521    8       **IF** *PCTS\_sem\_open* **THEN**  
522               **SETUP:** The call *sem\_open(name, oflag, mode, value )* creates a semaphore with an initial  
523               value of *value*.  
524           **ELSE NO\_OPTION**  
525           Conformance for *sem\_open*: *PASS, NO\_OPTION*

526    9       **IF** *PCTS\_sem\_open* **THEN**  
527               **TEST:** Valid initial values for semaphores are unsigned integers with values between 0 and  
528               {*SEM\_VALUE\_MAX*}, inclusive.  
529               **TR:** Test values of 0, {*SEM\_VALUE\_MAX*}, and {*SEM\_VALUE\_MAX*}+1, if it is less than  
530               {*UINT\_MAX*}.  
531           **ELSE NO\_OPTION**  
532           Conformance for *sem\_open*: *PASS, NO\_OPTION*

533    10       **IF** *PCTS\_sem\_open* **THEN**  
534               **TEST:** The user ID of the semaphore created by *sem\_open()* is the effective user ID of the  
535               process.  
536           **ELSE NO\_OPTION**  
537           Conformance for *sem\_open*: *PASS, NO\_OPTION*

538    11       **IF** *PCTS\_sem\_open* **THEN**  
539               **TEST:** The group ID of the semaphore created by *sem\_open()* is a system default group ID  
540               or the effective group ID of the process.  
541           **ELSE NO\_OPTION**  
542           Conformance for *sem\_open*: *PASS, NO\_OPTION*

543    12       **IF** *PCTS\_sem\_open* **THEN**  
544               **TEST:** The permission bits of the semaphore created by *sem\_open(name, oflag, mode, value*  
545               *)* are set to the value of the *mode* argument, except those set in the file mode creation  
546               mask of the process.  
547           **ELSE NO\_OPTION**  
548           Conformance for *sem\_open*: *PASS, NO\_OPTION*

549    **D\_1 IF** *PCTS\_sem\_open* and a PCD.1b documents the following **THEN**  
550               **TEST:** A PCD.1b that documents the effect of calling *sem\_open (name, oflag, mode, value*  
551               *)* with bits specified in *mode* other than the file permission bits does so in §11.2.3.2.  
552           **ELSE NO\_OPTION**  
553           Conformance for *sem\_open*: *PASS, NO\_OPTION*

554    13       **IF** *PCTS\_sem\_open* **THEN**

555                   **SETUP:** Create a new named semaphore with the call *sem\_open(name, oflag, ... )* and the  
556                                   O\_CREAT flag set in *oflag*.  
557                   **TEST:** Other processes can connect to the semaphore by calling *sem\_open()* with the same  
558                                   value of *name*.  
559                   **ELSE NO\_OPTION**  
560                   Conformance for *sem\_open*: PASS, NO\_OPTION

561   **14**           **IF PCTS\_sem\_open THEN**  
562                   **TEST:** When the semaphore *name* exists, the call *sem\_open (name, oflag, ... )* with the O\_CREAT  
563                                   and O\_EXCL flags set in *oflag*, fails.  
564                   **ELSE NO\_OPTION**  
565                   Conformance for *sem\_open*: PASS, NO\_OPTION

566   **15**           **IF PCTS\_sem\_open THEN**  
567                   **TEST:** The check for the existence of the semaphore in a call to *sem\_open( )* with o\_excl and  
568                                   O\_CREAT flag set, and the creation of the semaphore if it does not exist, are atomic  
569                                   with respect to other processes executing *sem\_open()* with O\_EXCL and O\_CREAT set.  
570                   **NOTE:** There is no known reliable test method for this assertion.  
571                   **ELSE NO\_OPTION**  
572                   Conformance for *sem\_open*: PASS, NO\_TEST, NO\_OPTION

573   **D\_2** **IF PCTS\_sem\_open** and a PCD.1b documents the following **THEN**  
574                   **TEST:** A PCD.1b that documents the effect of calling *sem\_open()* with O\_EXCL set and  
575                                   O\_CREAT not set does so in §11.2.3.2.  
576                   **ELSE NO\_OPTION**  
577                   Conformance for *sem\_open*: PASS, NO\_OPTION

578   **D\_3** **IF PCTS\_sem\_open** and a PCD.1b documents the following **THEN**  
579                   **TEST:** A PCD.1b that documents the effect of calling *sem\_open()* with *oflag* other than  
580                                   O\_EXCL and O\_CREAT specified in the *oflag* parameter, does so in §11.2.3.2.  
581                   **ELSE NO\_OPTION**  
582                   Conformance for *sem\_open*: PASS, NO\_OPTION

583   **D\_4** **IF PCTS\_sem\_open** and a PCD.1b documents the following **THEN**  
584                   **TEST:** A PCD.1b that documents whether *name* appears in the filesystem does so in  
585                                   §11.2.3.2.  
586                   **ELSE NO\_OPTION**  
587                   Conformance for *sem\_open*: PASS, NO\_OPTION

588   **D\_5** **IF PCTS\_sem\_open** and a PCD.1b documents the following **THEN**  
589                   **TEST:** A PCD.1b that documents whether *name* is visible to functions that take pathnames  
590                                   as arguments does so in §11.2.3.2.  
591                   **ELSE NO\_OPTION**  
592                   Conformance for *sem\_open*: PASS, NO\_OPTION

593   **16**           **M\_GA\_portableFileNames(sem\_open)**  
594                   **SEE:** Assertion GA\_portableFileNames in §2.2.2.40  
595                   Conformance for *sem\_open*: PASS, NO\_OPTION

596   **17**           **M\_GA\_upperLowerNames(sem\_open)**  
597                   **SEE:** Assertion GA\_upperLowerNames in §2.2.2.40  
598                   Conformance for *sem\_open*: PASS, NO\_OPTION

599   **18**           **M\_GA\_PRNoTrunc(sem\_open)**  
600                   **SEE:** Assertion GA\_PRNoTrunc in §2.3.6  
601                   Conformance for *sem\_open*: PASS, NO\_OPTION

602   **19**           **M\_GA\_PRNoTruncError(sem\_open)**  
603                   **SEE:** Assertion GA\_PRNoTruncError in §2.3.6

604 *Conformance for sem\_open: PASS, NO\_OPTION*

605 **20 IF PCTS\_sem\_open THEN**

606 **SETUP:** Call *sem\_open(name, oflag, ... )* where *name* begins with the slash character.

607 **TEST:** Processes calling *sem\_open()* with the same value of *name* refer to the same

608 semaphore object, as long as that name has not been removed.

609 **ELSE NO\_OPTION**

610 *Conformance for sem\_open: PASS, NO\_OPTION*

611 **D\_6 IF PCTS\_sem\_open THEN**

612 **TEST:** A PCD.1b documents the effect of calling *sem\_open(name, oflag, ... )* when *name*

613 does not begin with a slash character in §11.2.3.2.

614 **ELSE NO\_OPTION**

615 *Conformance for sem\_open: PASS, NO\_OPTION*

616 **D\_7 IF PCTS\_sem\_open THEN**

617 **TEST:** A PCD.1b documents the interpretation of slash characters in §11.2.3.2.

618 **ELSE NO\_OPTION**

619 *Conformance for sem\_open: PASS, NO\_OPTION*

620

621 **21 IF PCTS\_sem\_open THEN**

622 **SETUP:** Call *sem\_open(name, oflag, ... )* with a single value for *name* and no intervening

623 calls to *sem\_unlink()*.

624 **TEST:** The same semaphore address is returned for each successful call.

625 **ELSE NO\_OPTION**

626 *Conformance for sem\_open: PASS, NO\_OPTION*

627 **D\_8 IF PCTS\_sem\_open and a PCD.1b documents the following THEN**

628 **TEST:** A PCD.1b that documents the effects of references to copies of semaphores created

629 with *sem\_open()* does so in §11.2.3.2.

630 **ELSE NO\_OPTION**

631 *Conformance for sem\_open: PASS, NO\_OPTION*

632 **D\_9 IF PCTS\_sem\_open and a PCD.1b documents the following THEN**

633 **TEST:** A PCD.1b that whether or not it supports the *sem\_open()* function does so in

634 §11.2.3.2.

635 **ELSE NO\_OPTION**

636 *Conformance for sem\_open: PASS, NO\_OPTION*

637 **11.2.3.3 Returns**

638 **R\_1 IF PCTS\_sem\_open THEN**

639 **TEST:** When a call to *sem\_open ()* completes successfully, the address of the semaphore is

640 returned.

641 **ELSE NO\_OPTION**

642 **SEE:** Assertion *sem\_open* in §11.2.3.2

643 **11.2.3.4 Errors**

644 **22 IF PCTS\_sem\_open THEN**

645 **TEST:** A call to *sem\_open()* when the named semaphore exists and the permissions specified

646 by *oflag* are denied, or the named semaphore does not exist and permission to create

647 the named semaphore is denied, returns a value of -1 and sets *errno* to [EACCESS]

648 **ELSE NO\_OPTION**

649 *Conformance for sem\_open: PASS, NO\_OPTION*

650 **23 IF PCTS\_sem\_open THEN**

651                   **TEST:**    A call to *sem\_open()* when O\_CREAT and O\_EXCL are set and the named semaphore  
652                                    already exists, returns a value of -1 and sets *errno* to [EEXIST]  
653                   **ELSE NO\_OPTION**  
654                   *Conformance for sem\_open: PASS, NO\_OPTION*

655   **24**           **IF PCTS\_sem\_open THEN**  
656                   **TEST:**    A call to *sem\_open()* when the *sem\_open()* operation is interrupted by a signal returns  
657                                    a value of -1 and sets *errno* to [EINTR]  
658                   **ELSE NO\_OPTION**  
659                   *Conformance for sem\_open: PASS, NO\_OPTION*

660   **25**           **IF PCTS\_sem\_open THEN**  
661                   **TEST:**    A call to *sem\_open()* when the *sem\_open()* operation is not supported for the given  
662                                    name returns a value of -1 and sets *errno* to [EINVAL]  
663                   **NOTE:**    A subroutine is recommended that either returns a name for which *sem\_open()* is not  
664                                    supported or indicates that there is no way to generate *sem\_open()* on the system.  
665                   **ELSE NO\_OPTION**  
666                   *Conformance for sem\_open: PASS, NO\_OPTION*

667   **D\_10**          **IF PCTS\_sem\_open THEN**  
668                   **TEST:**    The PCD.1b documents under what circumstances [EINVAL] may be returned in  
669                                    §11.2.3.4.  
670                   **ELSE NO\_OPTION**  
671                   *Conformance for sem\_open: PASS, NO\_OPTION*

672   **26**           **IF PCTS\_sem\_open THEN**  
673                   **TEST:**    A call to *sem\_open(name, oflag, ... )* when O\_CREAT is specified in *oflag* and *value*  
674                                    is greater than {SEM\_VALUE\_MAX} returns a value of -1 and sets *errno* to [EINVAL]  
675                   **ELSE NO\_OPTION**  
676                   *Conformance for sem\_open: PASS, NO\_OPTION*

677   **27**           **IF PCTS\_sem\_open THEN**  
678                   **TEST:**    A call to *sem\_open()* when too many semaphore descriptors are currently in use by  
679                                    this process returns a value of -1 and sets *errno* to [EMFILE]  
680                   **ELSE NO\_OPTION**  
681                   *Conformance for sem\_open: PASS, NO\_OPTION*

682   **28**           **IF PCTS\_sem\_open THEN**  
683                   **IF PCTS\_SEM\_IS\_FD and ( {OPEN\_MAX}< PCTS\_OPEN\_MAX) THEN**  
684                    **TEST:**    A call to *sem\_open()* when too many file descriptors are currently in use by this  
685                                    process returns a value of -1 and sets *errno* to [EMFILE]  
686                    **TR:** Open {OPEN\_MAX} files. Try to open a semaphore.  
687                   **ELSE NO\_TEST\_SUPPORT**  
688                   **ELSE NO\_OPTION**  
689                   *Conformance for sem\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

690   **29**           **IF PCTS\_sem\_open THEN**  
691                   **IF {PATH\_MAX}<= PCTS\_PATH\_MAX THEN**  
692                    **TEST:**    A call to *sem\_open()* when the length of the *name* string exceeds {PATH\_MAX}  
693                                    returns a value of -1 and sets *errno* to [ENAMETOOLONG]  
694                    **TR:** Open {OPEN\_MAX} files. Try to open a semaphore.  
695                   **ELSE NO\_TEST\_SUPPORT**  
696                   **ELSE NO\_OPTION**  
697                   *Conformance for sem\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

698   **30**           **IF PCTS\_sem\_open THEN**  
699                   **IF {NAME\_MAX}<= PCTS\_NAME\_MAX and {\_POSIX\_NO\_TRUNC} THEN**

700                   **TEST:**     A call to *sem\_open()* when a pathname component is longer than {NAME\_MAX}  
701                                   while {\_POSIX\_NO\_TRUNC} is in effect returns a value of -1 and sets *errno* to  
702                                   [ENAMETOOLONG]  
703                   **ELSE NO\_TEST\_SUPPORT**  
704                   **ELSE NO\_OPTION**  
705                   *Conformance for sem\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

706   **31**           **IF PCTS\_sem\_open THEN**  
707                   **TEST:**     A call to *sem\_open()* when too many semaphores are currently open in the system  
708                                   returns a value of -1 and sets *errno* to [ENFILE]  
709                   **ELSE NO\_OPTION**  
710                   *Conformance for sem\_open: PASS, NO\_OPTION*

711   **32**           **IF PCTS\_sem\_open THEN**  
712                   **TEST:**     A call to *sem\_open()* when O\_CREAT is not set and the named semaphore does not  
713                                   exist returns a value of -1 and sets *errno* to [ENOENT]  
714                   **ELSE NO\_OPTION**  
715                   *Conformance for sem\_open: PASS, NO\_OPTION*

716   **33**           **IF PCTS\_sem\_open THEN**  
717                   **TEST:**     A call to *sem\_open()* when there is insufficient space for the creation of the new  
718                                   named semaphore returns a value of -1 and sets *errno* to [ENOSPC]  
719                   **NOTE:**     There is no known reliable test method for this assertion.  
720                   **ELSE NO\_OPTION**  
721                   *Conformance for sem\_open: PASS, NO\_TEST, NO\_OPTION*

722   **34**           **IF not PCTS\_sem\_open THEN**  
723                   **TEST:**     A call to *sem\_open()* returns a value of -1 and sets *errno* to [ENOSYS].  
724                   **ELSE NO\_OPTION**  
725                   *Conformance for sem\_open: PASS, NO\_OPTION*

#### 726   **11.2.4     Close a Named Semaphore**

727   Function: *sem\_close()*

##### 728   **11.2.4.1 Synopsis**

729   **1**  
730       *M\_GA\_stdC\_proto\_decl(int; sem\_close; sem\_t \*sem; semaphore.h;;)*  
731       **SEE:**     Assertion GA\_stdC\_proto\_decl in §2.7.3  
732       *Conformance for sem\_close: PASS[1, 2], NO\_OPTION*

733   **2**  
734       *M\_GA\_commonC\_int\_result\_decl(sem\_close; semaphore.h;;)*  
735       **SEE:**     Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
736       *Conformance for sem\_close: PASS[1, 2], NO\_OPTION*

737   **3**  
738       *M\_GA\_macro\_result\_decl(int; sem\_close; semaphore.h;;)*  
739       **SEE:**     Assertion GA\_macro\_result\_decl in §1.3.4  
740       *Conformance for sem\_close: PASS, NO\_OPTION*

741   **4**  
742       *M\_GA\_macro\_args ( sem\_close; semaphore.h;;)*  
743       **SEE:**     Assertion GA\_macro\_args in §2.7.3  
744       *Conformance for sem\_close: PASS, NO\_OPTION*

##### 745   **11.2.4.2 Description**



746 **sem\_close**  
 747 **IF** *PCTS\_sem\_close* **THEN**  
 748 **IF:** *PCTS\_sem\_open* and {SEM\_NSEMS\_MAX} < *PCTS\_SEM\_NSEMS\_MAX* **THEN**  
 749 **TEST:** The *sem\_close*() function makes any system resources allocated by the system  
 750 available for reuse by a subsequent *sem\_open*() call in this process and returns  
 751 zero.  
 752 **TR:** Open {SEM\_NSEMS\_MAX} semaphores, close one, then open another.  
 753 **ELSE NO\_TEST\_SUPPORT**  
 754 **ELSE NO\_OPTION**  
 755 *Conformance for sem\_close: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

756 **D\_1 IF** *PCTS\_sem\_close* and a PCD.1b documents the following **THEN**  
 757 **TEST:** A PCD.1b that documents the effects of calling *sem\_close*() for an unnamed  
 758 semaphore does so in §11.2.4.2.  
 759 **ELSE NO\_OPTION**  
 760 *Conformance for sem\_close: PASS, NO\_OPTION*

761 **D\_2 IF** *PCTS\_sem\_close* and a PCD.1b documents the following **THEN**  
 762 **TEST:** A PCD.1b that documents the effects subsequent use of the semaphore *sem* does so  
 763 in §11.2.4.2.  
 764 **ELSE NO\_OPTION**  
 765 *Conformance for sem\_close: PASS, NO\_OPTION*

766 **5 IF** *PCTS\_sem\_close* **THEN**  
 767 **TEST:** The *sem\_close*() has no effect on the state of the semaphore, if the semaphore has not  
 768 been removed with a successful call to *sem\_unlink*().  
 769 **ELSE NO\_OPTION**  
 770 *Conformance for sem\_close: PASS, NO\_OPTION*

771 **6 IF** *PCTS\_sem\_close* **THEN**  
 772 **IF:** *PCTS\_sem\_open* *PCTS\_sem\_unlink* **THEN**  
 773 **SETUP:** Call *sem\_open*() with O\_CREAT, then call *sem\_unlink*() successfully on the  
 774 same semaphore.  
 775 **TEST:** When all processes that have opened the semaphore close it, the semaphore is  
 776 no longer accessible.  
 777 **ELSE NO\_TEST\_SUPPORT**  
 778 **ELSE NO\_OPTION**  
 779 *Conformance for sem\_close: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

780 **D\_3 IF** *PCTS\_sem\_close* and a PCD.1b documents the following **THEN**  
 781 **TEST:** A PCD.1b that documents whether or not it supports the *sem\_close*() function does so  
 782 in §11.2.4.2.  
 783 **ELSE NO\_OPTION**  
 784 *Conformance for sem\_close: PASS, NO\_OPTION*

### 785 11.2.4.3 Returns

786 **R\_1 IF** *PCTS\_sem\_close* **THEN**  
 787 **TEST:** When a call to *sem\_close*() completes successfully, the interface returns a value of  
 788 0.  
 789 **ELSE NO\_OPTION**  
 790 **SEE:** Assertion *sem\_close* in §11.2.4.2

791 **R\_2 IF** *PCTS\_sem\_close* **THEN**  
 792 **TEST:** When a call to *sem\_close*() completes unsuccessfully, the interface returns a value  
 793 of -1 and sets *errno* to indicate the error.  
 794 **ELSE NO\_OPTION**  
 795 **SEE:** Assertion *sem\_close* in §11.2.4.4

796 **11.2.4.4 Errors**

797 **7** **IF** *PCTS\_sem\_close* **THEN**  
 798 **TEST:** A call to *sem\_close()*, when argument is not a valid semaphore descriptor, returns a  
 799 value of -1 and sets *errno* to {EINVAL}  
 800 **NOTE:** A subroutine is recommended that either returns an invalid semaphore descriptor or  
 801 indicates that there is no way to generate an invalid semaphore descriptor on the  
 802 system.  
 803 **ELSE** *NO\_OPTION*  
 804 *Conformance for sem\_close: PASS, NO\_OPTION*

805 **8** **IF** not *PCTS\_sem\_close* **THEN**  
 806 **TEST:** A call to *sem\_close()* returns a value of -1 and sets *errno* to [ENOSYS].  
 807 **ELSE** *NO\_OPTION*  
 808 *Conformance for sem\_close: PASS, NO\_OPTION*

809 **11.2.5 Remove a Named Semaphore**

810 Function: *sem\_unlink()*

811 **11.2.5.1 Synopsis**

812 **1**  
 813 *M\_GA\_stdC\_proto\_decl(int; sem\_unlink; const char \*name; semaphore.h;;;)*  
 814 **SEE:** Assertion *GA\_stdC\_proto\_decl* in §2.7.3  
 815 *Conformance for sem\_unlink: PASS[1, 2], NO\_OPTION*

816 **2**  
 817 *M\_GA\_commonC\_int\_result\_decl(sem\_unlink; semaphore.h;;;)*  
 818 **SEE:** Assertion *GA\_commonC\_int\_result\_decl* in §2.7.3  
 819 *Conformance for sem\_unlink: PASS[1, 2], NO\_OPTION*

820 **3**  
 821 *M\_GA\_macro\_result\_decl(int; sem\_unlink; semaphore.h;;;)*  
 822 **SEE:** Assertion *GA\_macro\_result\_decl* in §1.3.4  
 823 *Conformance for sem\_unlink: PASS, NO\_OPTION*

824 **4**  
 825 *M\_GA\_macro\_args ( sem\_unlink; semaphore.h;;;)*  
 826 **SEE:** Assertion *GA\_macro\_args* in §2.7.3  
 827 *Conformance for sem\_unlink: PASS, NO\_OPTION*

828 **11.2.5.2 Description**

829 **sem\_unlink**  
 830 **IF** *PCTS\_sem\_unlink* **THEN**  
 831 **TEST:** The call *sem\_unlink(name)* removes the semaphore named by the string *name*, and  
 832 returns the value zero.  
 833 **ELSE** *NO\_OPTION*  
 834 *Conformance for sem\_unlink: PASS, NO\_OPTION*

835 **5** **IF** *PCTS\_sem\_unlink* **THEN**  
 836 **TEST:** When the semaphore named by *name* is currently referenced by other processes, then  
 837 *sem\_unlink(name)* has no effect on the state of the semaphore.  
 838 **ELSE** *NO\_OPTION*  
 839 *Conformance for sem\_unlink: PASS, NO\_OPTION*

840 **6** **FOR:** *sem\_close()*, *\_exit()*, *execl()*, *execv()*, *execle()*, *execve()*, *execp()*, and *execvp()*

841           **IF** *PCTS\_sem\_unlink* **THEN**  
842               **TEST:**    When one or more processes has the semaphore open when *sem\_unlink()* is called,  
843                            destruction of the semaphore is postponed until all references to the semaphore have  
844                            been destroyed by a call to *function()*.  
845               **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
846                            assertion is to be read by substituting *function()* with the current function specified  
847                            in the FOR clause. The name of the function also is to be substituted for each  
848                            occurrence in the construct *PCTS\_function*.  
849           **ELSE NO\_OPTION**  
850           *Conformance for sem\_unlink: PASS, NO\_OPTION*

851    **7**       **IF** *PCTS\_sem\_unlink* **THEN**  
852               **IF** *PCTS\_sem\_open*  
853               **TEST:**    Calls to *sem\_open()* to re-create or re-connect to the semaphore refer to a new  
854                            semaphore after *sem\_unlink()* is called.  
855               **ELSE NO\_TEST\_SUPPORT**  
856           **ELSE NO\_OPTION**  
857           *Conformance for sem\_unlink: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

858    **8**       **IF** *PCTS\_sem\_unlink* **THEN**  
859               **TEST:**    When any process references name, the call *sem\_unlink(name)* returns immediately.  
860           **ELSE NO\_OPTION**  
861           *Conformance for sem\_unlink: PASS, NO\_OPTION*

862    **D\_1** **IF** *PCTS\_sem\_unlink* and a PCD.1b documents the following **THEN**  
863               **TEST:**    A PCD.1b that documents whether or not it supports the *sem\_unlink()* function does  
864                            so in §11.2.5.2.  
865           **ELSE NO\_OPTION**  
866           *Conformance for sem\_unlink: PASS, NO\_OPTION*

867    **11.2.5.3 Returns**

868    **R\_1** **IF** *PCTS\_sem\_unlink* **THEN**  
869               **TEST:**    When a call to *sem\_unlink()* completes successfully, the interface returns a value of  
870                            0.  
871           **ELSE NO\_OPTION**  
872           **SEE:**        Assertion *sem\_unlink* in §11.2.5.2

873    **R\_2** **IF** *PCTS\_sem\_unlink* **THEN**  
874               **TEST:**    When a call to *sem\_unlink()* completes unsuccessfully, the interface returns a value  
875                            of -1, sets *errno* to indicate the error, and does not change the semaphore.  
876           **ELSE NO\_OPTION**  
877           **SEE:**        All assertions in §11.2.5.4

878    **11.2.5.4 Errors**

879    **9**       **IF** *PCTS\_sem\_unlink* **THEN**  
880               **TEST:**    A call to *sem\_unlink()*, when permission is denied to unlink the named semaphore,  
881                            returns a value of -1 and sets *errno* to [EACCESS].  
882           **ELSE NO\_OPTION**  
883           *Conformance for sem\_unlink: PASS, NO\_OPTION*

884    **10**      **IF** *PCTS\_sem\_unlink* **THEN**  
885               **IF** *{\_posix\_no\_trunc}* AND *{name\_max}* <= *PCTS\_NAME\_MAX* **THEN**  
886               **TEST:**    A call to *sem\_unlink()*, when the length of the *name* string exceeds {NAME\_MAX}  
887                            while {\_POSIX\_NO\_TRUNC} is in effect, returns a value of -1 and sets *errno* to  
888                            [ENAMETOOLONG]  
889               **ELSE NO\_TEST\_SUPPORT**  
890           **ELSE NO\_OPTION**

891 *Conformance for sem\_unlink: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

892 **11 IF PCTS\_sem\_unlink THEN**

893 **TEST:** A calls to *sem\_unlink()*, when the named semaphore does not exist, returns a value

894 of -1 and sets *errno* to [ENOENT].

895 **NOTE:** A subroutine is recommended that either returns the name of a semaphore that does

896 not exist or indicates that there is no way to generate the name of a semaphore that

897 does not exist in the system.

898 **ELSE NO\_OPTION**

899 *Conformance for sem\_unlink: PASS, NO\_OPTION*

900 **12 IF not PCTS\_sem\_unlink THEN**

901 **TEST:** A calls to *sem\_unlink()* returns a value of -1 and sets *errno* to [ENOSYS].

902 **ELSE NO\_OPTION**

903 *Conformance for sem\_unlink: PASS, NO\_OPTION*

## 904 **11.2.6 Lock a Semaphore**

905 Functions: *sem\_wait()*, *sem\_trywait()*

### 906 **11.2.6.1 Synopsis**

907 **1**

908 *M\_GA\_stdC\_proto\_decl(int; sem\_wait; sem\_t \*sem; semaphore.h;;)*

909 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3

910 *Conformance for sem\_wait: PASS[1, 2], NO\_OPTION*

911 **2**

912 *M\_GA\_commonC\_int\_result\_decl(sem\_wait; semaphore.h;;)*

913 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3

914 *Conformance for sem\_wait: PASS[1, 2], NO\_OPTION*

915 **3**

916 *M\_GA\_macro\_result\_decl(int; sem\_wait; semaphore.h;;)*

917 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4

918 *Conformance for sem\_wait: PASS, NO\_OPTION*

919 **4**

920 *M\_GA\_macro\_args ( sem\_wait; semaphore.h;;)*

921 **SEE:** Assertion GA\_macro\_args in §2.7.3

922 *Conformance for sem\_wait: PASS, NO\_OPTION*

923 **5**

924 *M\_GA\_stdC\_proto\_decl(int; sem\_trywait; sem\_t \*sem; semaphore.h;;)*

925 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3

926 *Conformance for sem\_trywait: PASS[5, 6], NO\_OPTION*

927 **6**

928 *M\_GA\_commonC\_int\_result\_decl(sem\_trywait; sem\_t \*sem; semaphore.h;;)*

929 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3

930 *Conformance for sem\_trywait: PASS[5, 6], NO\_OPTION*

931 **7**

932 *M\_GA\_macro\_result\_decl(int; sem\_trywait; semaphore.h;;)*

933 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4

934 *Conformance for sem\_trywait: PASS, NO\_OPTION*

935 **8**

936 *M\_GA\_macro\_args ( sem\_trywait; semaphore.h;;)*  
 937 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 938 *Conformance for sem\_trywait: PASS, NO\_OPTION*

### 939 11.2.6.2 Description

#### 940 **sem\_wait**

941 **FOR:** *sem\_init()* and *sem\_open()*  
 942 **IF** *PCTS\_sem\_wait* **THEN**  
 943 **IF** *PCTS\_function* **THEN**  
 944 **SETUP:** Create a semaphore using *function()*.  
 945 **TEST:** When the call *sem\_wait(sem)* locks the semaphore *sem*, with the semaphore  
 946 lock operation, it returns the value zero.  
 947 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 948 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 949 not a way to get appropriate privilege to call *sem\_init()*.  
 950 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
 951 assertion is to be read by substituting *function()* with the current function  
 952 specified in the FOR clause. The name of the function also is to be substituted  
 953 for each occurrence in the construct *PCTS\_function*.  
 954 **ELSE** *NO\_TEST\_SUPPORT*  
 955 **ELSE** *NO\_OPTION*  
 956 *Conformance for sem\_wait: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

#### 957 **sem\_trywait**

958 **FOR:** *sem\_init()* and *sem\_open()*  
 959 **IF** *PCTS\_sem\_trywait* **THEN**  
 960 **IF** *PCTS\_function* **THEN**  
 961 **SETUP:** Create a semaphore using *function()*.  
 962 **TEST:** When the call *sem\_trywait(sem)* locks the semaphore *sem*, with the semaphore  
 963 lock operation, it returns the value zero.  
 964 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 965 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 966 not a way to get appropriate privilege to call *sem\_init()*.  
 967 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
 968 assertion is to be read by substituting *function()* with the current function  
 969 specified in the FOR clause. The name of the function also is to be substituted  
 970 for each occurrence in the construct *PCTS\_function*.  
 971 **ELSE** *NO\_TEST\_SUPPORT*  
 972 **ELSE** *NO\_OPTION*  
 973 *Conformance for sem\_trywait: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

974 **9** **FOR:** *sem\_init()* and *sem\_open()*  
 975 **IF** *PCTS\_sem\_wait* **THEN**  
 976 **IF** *PCTS\_function* **THEN**  
 977 **SETUP:** Create a semaphore using *function()*.  
 978 **TEST:** When the semaphore value is currently zero, and the call to *sem\_wait(sem)* is  
 979 not interrupted by a signal, when the call returns the semaphore referenced by  
 980 *sem* is locked.  
 981 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 982 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 983 not a way to get appropriate privilege to call *sem\_init()*.  
 984 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
 985 assertion is to be read by substituting *function()* with the current function  
 986 specified in the FOR clause. The name of the function also is to be substituted  
 987 for each occurrence in the construct *PCTS\_function*.  
 988 **ELSE** *NO\_TEST\_SUPPORT*  
 989 **ELSE** *NO\_OPTION*  
 990 *Conformance for sem\_wait: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

991     **10**     **FOR:**     *sem\_init()* and *sem\_open()*  
992     **IF** *PCTS\_sem\_trywait* **THEN**  
993         **IF** *PCTS\_function* **THEN**  
994             **SETUP:**   Create a semaphore using *function()*.  
995             **TEST:**     When the semaphore value is currently positive, and the call to *sem\_wait(sem)*  
996                         is not interrupted by a signal, when the call returns the semaphore referenced  
997                         by *sem* is locked, and the interface returns the value zero.  
998             **TR:**     When testing for *sem\_init()*, perform the test consistent with the flag  
999                         *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1000                         not a way to get appropriate privilege to call *sem\_init()*.  
1001             **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
1002                         assertion is to be read by substituting *function()* with the current function  
1003                         specified in the FOR clause. The name of the function also is to be substituted  
1004                         for each occurrence in the construct *PCTS\_function*.  
1005             **ELSE** *NO\_TEST\_SUPPORT*  
1006         **ELSE** *NO\_OPTION*  
1007         Conformance for *sem\_trywait*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1008     **11**     **FOR:**     *sem\_init()* and *sem\_open()*  
1009     **IF** *PCTS\_sem\_trywait* **THEN**  
1010         **IF** *PCTS\_function* **THEN**  
1011             **SETUP:**   Create a semaphore using *function()*.  
1012             **TEST:**     When the semaphore value is currently zero, the call to *sem\_trywait()* does not  
1013                         lock the semaphore referenced by *sem*.  
1014             **TR:**     When testing for *sem\_init()*, perform the test consistent with the flag  
1015                         *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1016                         not a way to get appropriate privilege to call *sem\_init()*.  
1017             **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
1018                         assertion is to be read by substituting *function()* with the current function  
1019                         specified in the FOR clause. The name of the function also is to be substituted  
1020                         for each occurrence in the construct *PCTS\_function*.  
1021             **ELSE** *NO\_TEST\_SUPPORT*  
1022         **ELSE** *NO\_OPTION*  
1023         Conformance for *sem\_trywait*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1024     **12**     **FOR:**     *sem\_init()* and *sem\_open()*  
1025     **IF** *PCTS\_sem\_wait* **THEN**  
1026         **IF** *PCTS\_function* and *sem\_post()* **THEN**  
1027             **SETUP:**   Create a semaphore using *function()*.  
1028             **TEST:**     After a successful *sem\_wait()* call, the semaphore is locked and remains  
1029                         locked until the *sem\_post()* function is executed and returns successfully.  
1030             **TR:**     When testing for *sem\_init()*, perform the test consistent with the flag  
1031                         *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1032                         not a way to get appropriate privilege to call *sem\_init()*.  
1033             **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
1034                         assertion is to be read by substituting *function()* with the current function  
1035                         specified in the FOR clause. The name of the function also is to be substituted  
1036                         for each occurrence in the construct *PCTS\_function*.  
1037             **ELSE** *NO\_TEST\_SUPPORT*  
1038         **ELSE** *NO\_OPTION*  
1039         Conformance for *sem\_wait*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1040     **13**     **FOR:**     *sem\_init()* and *sem\_open()*  
1041     **IF** *PCTS\_sem\_trywait* **THEN**  
1042         **IF** *PCTS\_function* and *sem\_post()* **THEN**  
1043             **SETUP:**   Create a semaphore using *function()*.  
1044             **TEST:**     After a successful call to *sem\_trywait()*, the semaphore is locked and remains  
1045                         locked until the *sem\_post* function is executed and returns successfully.

1046                   **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1047                    *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1048                    not a way to get appropriate privilege to call *sem\_init()*.  
1049                   **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
1050                    assertion is to be read by substituting *function()* with the current function  
1051                    specified in the FOR clause. The name of the function also is to be substituted  
1052                    for each occurrence in the construct *PCTS\_function*.  
1053                    **ELSE NO\_TEST\_SUPPORT**  
1054                    **ELSE NO\_OPTION**  
1055                    *Conformance for sem\_trywait: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1056    **14**       **FOR:**     *sem\_init()* and *sem\_open()*  
1057                    **IF** *PCTS\_sem\_wait* **THEN**  
1058                    **IF** *PCTS\_function* **THEN**  
1059                    **SETUP:** Create a semaphore using *function()*.  
1060                    **TEST:** The *sem\_wait()* function is interruptible by the delivery of a signal.  
1061                    **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1062                    *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1063                    not a way to get appropriate privilege to call *sem\_init()*.  
1064                    **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
1065                    assertion is to be read by substituting *function()* with the current function  
1066                    specified in the FOR clause. The name of the function also is to be substituted  
1067                    for each occurrence in the construct *PCTS\_function*.  
1068                    **ELSE NO\_TEST\_SUPPORT**  
1069                    **ELSE NO\_OPTION**  
1070                    *Conformance for sem\_wait: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1071    **D\_1** **IF** *PCTS\_sem\_wait* and a PCD.1b documents the following **THEN**  
1072                    **TEST:** A PCD.1b that documents whether or not it supports the *sem\_wait()* function does so  
1073                    in §11.2.6.2.  
1074                    **ELSE NO\_OPTION**  
1075                    *Conformance for sem\_wait: PASS, NO\_OPTION*

1076    **D\_2** **IF** *PCTS\_sem\_trywait* and a PCD.1b documents the following **THEN**  
1077                    **TEST:** A PCD.1b that documents whether or not it supports the *sem\_trywait()* function does  
1078                    so in §11.2.6.2.  
1079                    **ELSE NO\_OPTION**  
1080                    *Conformance for sem\_trywait: PASS, NO\_OPTION*

1081

1082    **11.2.6.3 Returns**

1083    **15**       **FOR:**     *sem\_init()* and *sem\_open()*  
1084                    **IF** *PCTS\_sem\_wait* **THEN**  
1085                    **IF** *PCTS\_function* **THEN**  
1086                    **SETUP:** Create a semaphore using *function()*.  
1087                    **TEST:** The *sem\_wait(sem)* function returns 0 if the calling process successfully  
1088                    performed the semaphore lock operation on the semaphore designated by *sem*.  
1089                    **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1090                    *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1091                    not a way to get appropriate privilege to call *sem\_init()*.  
1092                    **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
1093                    assertion is to be read by substituting *function()* with the current function  
1094                    specified in the FOR clause. The name of the function also is to be substituted  
1095                    for each occurrence in the construct *PCTS\_function*.  
1096                    **ELSE NO\_TEST\_SUPPORT**  
1097                    **ELSE NO\_OPTION**  
1098                    *Conformance for sem\_wait: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1099 **16 FOR:** *sem\_init()* and *sem\_open()*  
1100 **IF** *PCTS\_sem\_trywait* **THEN**  
1101 **IF** *PCTS\_function* **THEN**  
1102 **SETUP:** Create a semaphore using *function()*.  
1103 **TEST:** The *sem\_trywait(sem)* function returns 0 if the calling process successfully  
1104 performed the semaphore lock operation on the semaphore designated by *sem*.  
1105 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1106 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1107 not a way to get appropriate privilege to call *sem\_init()*.  
1108 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
1109 assertion is to be read by substituting *function()* with the current function  
1110 specified in the FOR clause. The name of the function also is to be substituted  
1111 for each occurrence in the construct *PCTS\_function*.  
1112 **ELSE** *NO\_TEST\_SUPPORT*  
1113 **ELSE** *NO\_OPTION*  
1114 *Conformance for sem\_trywait: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1115 **R\_1 FOR:** *sem\_init()* and *sem\_open()*  
1116 **IF** *PCTS\_sem\_wait* **THEN**  
1117 **IF** *PCTS\_function* **THEN**  
1118 **SETUP:** Create a semaphore using *function()*.  
1119 **TEST:** When a call to *sem\_wait(sem)* completes successfully, the interface returns a  
1120 value of 0 and the semaphore designated by *sem* is locked by the semaphore  
1121 lock operation.  
1122 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1123 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1124 not a way to get appropriate privilege to call *sem\_init()*.  
1125 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
1126 assertion is to be read by substituting *function()* with the current function  
1127 specified in the FOR clause. The name of the function also is to be substituted  
1128 for each occurrence in the construct *PCTS\_function*.  
1129 **ELSE** *NO\_TEST\_SUPPORT*  
1130 **ELSE** *NO\_OPTION*  
1131 **SEE:** Assertion *sem\_wait* in §11.2.6.2

1132 **R\_2 FOR:** *sem\_init()* and *sem\_open()*  
1133 **IF** *PCTS\_sem\_trywait* **THEN**  
1134 **IF** *PCTS\_function* **THEN**  
1135 **SETUP:** Create a semaphore using *function()*.  
1136 **TEST:** When a call to *sem\_trywait(sem)* completes successfully, the interface returns  
1137 a value of 0 and the semaphore designated by *sem* is locked by the semaphore  
1138 lock operation.  
1139 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1140 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1141 not a way to get appropriate privilege to call *sem\_init()*.  
1142 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
1143 assertion is to be read by substituting *function()* with the current function  
1144 specified in the FOR clause. The name of the function also is to be substituted  
1145 for each occurrence in the construct *PCTS\_function*.  
1146 **ELSE** *NO\_TEST\_SUPPORT*  
1147 **ELSE** *NO\_OPTION*  
1148 **SEE:** Assertion *sem\_trywait* in §11.2.6.2

1149 **R\_3 FOR:** *sem\_init()* and *sem\_open()*  
1150 **IF** *PCTS\_sem\_wait* **THEN**  
1151 **IF** *PCTS\_function* **THEN**  
1152 **SETUP:** Create a semaphore using *function()*.



1153                   **TEST:**    When a call to *sem\_wait(sem)* completes unsuccessfully, the interface returns  
 1154   a value of -1, sets *errno* to indicate the error, and does not change the state of  
 1155   the semaphore.  
 1156                   **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 1157   *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1158   not a way to get appropriate privilege to call *sem\_init()*.  
 1159                   **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
 1160   assertion is to be read by substituting *function()* with the current function  
 1161   specified in the FOR clause. The name of the function also is to be substituted  
 1162   for each occurrence in the construct *PCTS\_function*.  
 1163   **ELSE NO\_TEST\_SUPPORT**  
 1164                   **ELSE NO\_OPTION**  
 1165                   **SEE:**     All assertions in §11.2.6.4 controlled by *PCTS\_sem\_wait*

1166   **R\_4 FOR:**    *sem\_init()* and *sem\_open()*  
 1167                   **IF PCTS\_sem\_trywait THEN**  
 1168   **IF PCTS\_function THEN**  
 1169   **SETUP:**    Create a semaphore using *function()*.  
 1170   **TEST:**     When a call to *sem\_trywait()* completes unsuccessfully, the interface returns  
 1171   a value of -1, sets *errno* to indicate the error, and does not change the state of  
 1172   the semaphore.  
 1173   **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 1174   *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1175   not a way to get appropriate privilege to call *sem\_init()*.  
 1176   **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
 1177   assertion is to be read by substituting *function()* with the current function  
 1178   specified in the FOR clause. The name of the function also is to be substituted  
 1179   for each occurrence in the construct *PCTS\_function*.  
 1180   **ELSE NO\_TEST\_SUPPORT**  
 1181   **ELSE NO\_OPTION**  
 1182   **SEE:**     All assertions in §11.2.6.4 controlled by *PCTS\_sem\_trywait*

#### 1183   **11.2.6.4 Errors**

1184   **17    FOR:**    *sem\_init()* and *sem\_open()*  
 1185                   **IF PCTS\_sem\_wait THEN**  
 1186   **IF PCTS\_function THEN**  
 1187   **TEST:**     A call to *sem\_wait()*, when the semaphore is already locked, so it cannot be  
 1188   immediately locked by the *sem\_trywait()* operation, returns a value of -1 and  
 1189   sets *errno* to [EAGAIN].  
 1190   **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 1191   *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1192   not a way to get appropriate privilege to call *sem\_init()*.  
 1193   **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
 1194   assertion is to be read by substituting *function()* with the current function  
 1195   specified in the FOR clause. The name of the function also is to be substituted  
 1196   for each occurrence in the construct *PCTS\_function*.  
 1197   **ELSE NO\_TEST\_SUPPORT**  
 1198   **ELSE NO\_OPTION**  
 1199   Conformance for *sem\_trywait*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1200   **18    FOR:**    *sem\_init()* and *sem\_open()*  
 1201                   **IF PCTS\_sem\_trywait THEN**  
 1202   **IF PCTS\_function THEN**  
 1203   **SETUP:**    Create a semaphore using *function()*.  
 1204   **TEST:**     A call to *sem\_trywait()*, when the semaphore is already locked, so it cannot be  
 1205   immediately locked by the *sem\_trywait()* operation, returns a value of -1 and  
 1206   sets *errno* to [EAGAIN].

1207                   **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1208                    *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1209                    not a way to get appropriate privilege to call *sem\_init()*.  
1210                   **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
1211                    assertion is to be read by substituting *function()* with the current function  
1212                    specified in the FOR clause. The name of the function also is to be substituted  
1213                    for each occurrence in the construct *PCTS\_function*.  
1214                    **ELSE NO\_TEST\_SUPPORT**  
1215                    **ELSE NO\_OPTION**  
1216                    Conformance for *sem\_trywait*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1217   **19**           **FOR:**     *sem\_init()* and *sem\_open()*  
1218                   **IF** *PCTS\_sem\_wait* **THEN**  
1219                    **IF** *PCTS\_function* **THEN**  
1220                      **SETUP:** Create a semaphore using *function()*.  
1221                      **TEST:** A call to *sem\_wait()*, when the *sem* argument does not refer to a valid  
1222                      semaphore returns a value of -1 and sets *errno* to [EINVAL].  
1223                      **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1224                      *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1225                      not a way to get appropriate privilege to call *sem\_init()*.  
1226                      **NOTE:** A subroutine is recommended that either returns an invalid semaphore or  
1227                      indicates that there is no way to generate an invalid semaphore on the system.  
  
1228                      The assertion is tested once for each function specified in the FOR clause. The  
1229                      assertion is to be read by substituting *function()* with the current function  
1230                      specified in the FOR clause. The name of the function also is to be substituted  
1231                      for each occurrence in the construct *PCTS\_function*.  
1232                      **ELSE NO\_TEST\_SUPPORT**  
1233                      **ELSE NO\_OPTION**  
1234                      Conformance for *sem\_wait*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1235   **20**           **FOR:**     *sem\_init()* and *sem\_open()*  
1236                   **IF** *PCTS\_sem\_trywait* **THEN**  
1237                    **IF** *PCTS\_function* **THEN**  
1238                      **SETUP:** Create a semaphore using *function()*.  
1239                      **TEST:** A call to *sem\_trywait(sem)*, when the *sem* argument does not refer to a valid  
1240                      semaphore returns a value of -1 and sets *errno* to [EINVAL].  
1241                      **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1242                      *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1243                      not a way to get appropriate privilege to call *sem\_init()*.  
1244                      **NOTE:** A subroutine is recommended that either returns an invalid semaphore or  
1245                      indicates that there is no way to generate an invalid semaphore on the system.  
  
1246                      The assertion is tested once for each function specified in the FOR clause. The  
1247                      assertion is to be read by substituting *function()* with the current function  
1248                      specified in the FOR clause. The name of the function also is to be substituted  
1249                      for each occurrence in the construct *PCTS\_function*.  
1250                      **ELSE NO\_TEST\_SUPPORT**  
1251                      **ELSE NO\_OPTION**  
1252                      Conformance for *sem\_trywait*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1253   **21**           **FOR:**     *sem\_init()* and *sem\_open()*  
1254                   **IF** *PCTS\_sem\_wait* **THEN**  
1255                    **IF** *PCTS\_function* **THEN**  
1256                      **SETUP:** Create a semaphore using *function()*.  
1257                      **TEST:** A call to *sem\_wait()*, interrupted by a signal, returns a value of -1 and sets  
1258                      *errno* to [EINTR].

1259                   **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1260                   *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1261                   not a way to get appropriate privilege to call *sem\_init()*.  
1262                   **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
1263                   assertion is to be read by substituting *function()* with the current function  
1264                   specified in the FOR clause. The name of the function also is to be substituted  
1265                   for each occurrence in the construct *PCTS\_function*.  
1266                   **ELSE NO\_TEST\_SUPPORT**  
1267                   **ELSE NO\_OPTION**  
1268                   *Conformance for sem\_trywait: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1269    **22**           **FOR:**     *sem\_init()* and *sem\_open()*  
1270                   **IF** *PCTS\_sem\_trywait* **THEN**  
1271                    **IF** *PCTS\_function* **THEN**  
1272                      **SETUP:** Create a semaphore using *function()*.  
1273                      **TEST:**    A call to *sem\_trywait()*, when interrupted by a signal, returns a value of -1 and  
1274                      sets *errno* to [EINVAL].  
1275                      **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1276                      *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1277                      not a way to get appropriate privilege to call *sem\_init()*.  
1278                      **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
1279                      assertion is to be read by substituting *function()* with the current function  
1280                      specified in the FOR clause. The name of the function also is to be substituted  
1281                      for each occurrence in the construct *PCTS\_function*.  
1282                      **ELSE NO\_TEST\_SUPPORT**  
1283                      **ELSE NO\_OPTION**  
1284                      *Conformance for sem\_trywait: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1285    **23**           **IF** not *PCTS\_sem\_wait* **THEN**  
1286                      **TEST:**    A call to *sem\_wait()* returns a value of -1 and sets *errno* to [ENOSYS].  
1287                      **ELSE NO\_OPTION**  
1288                      *Conformance for sem\_wait: PASS, NO\_OPTION*

1289    **24**           **IF** not *PCTS\_sem\_trywait* **THEN**  
1290                      **TEST:**    A call to *sem\_trywait()* returns a value of -1 and sets *errno* to [ENOSYS].  
1291                      **ELSE NO\_OPTION**  
1292                      *Conformance for sem\_trywait: PASS, NO\_OPTION*

1293    **25**           **FOR:**     *sem\_init()* and *sem\_open()*  
1294                   **IF** *PCTS\_sem\_wait* **THEN**  
1295                    **IF** *PCTS\_function* **THEN**  
1296                      **SETUP:** Create a semaphore using *function()*.  
1297                      **TEST:**    A call to *sem\_wait()*, when a deadlock condition is detected, returns a value of  
1298                      -1 and sets *errno* to [EDEADLK].  
1299                      **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1300                      *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1301                      not a way to get appropriate privilege to call *sem\_init()*.  
1302                      **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
1303                      assertion is to be read by substituting *function()* with the current function  
1304                      specified in the FOR clause. The name of the function also is to be substituted  
1305                      for each occurrence in the construct *PCTS\_function*.  
1306                      **ELSE NO\_TEST\_SUPPORT**  
1307                      **ELSE NO\_OPTION**  
1308                      *Conformance for sem\_wait: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1309    **26**           **FOR:**     *sem\_init()* and *sem\_open()*  
1310                   **IF** *PCTS\_sem\_trywait* **THEN**  
1311                    **IF** *PCTS\_function* **THEN**  
1312                      **SETUP:** Create a semaphore using *function()*.

1313                   **TEST:**    A call to *sem\_trywait()*, when a deadlock condition is detected, returns a value  
 1314   of -1 and sets *errno* to [EDEADLK].  
 1315                   **TR:**    When testing for *sem\_init()*, perform the test consistent with the flag  
 1316   *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1317   not a way to get appropriate privilege to call *sem\_init()*.  
 1318                   **NOTE:**   The assertion is tested once for each function specified in the FOR clause. The  
 1319   assertion is to be read by substituting *function()* with the current function  
 1320   specified in the FOR clause. The name of the function also is to be substituted  
 1321   for each occurrence in the construct *PCTS\_function*.  
 1322                   **ELSE NO\_TEST\_SUPPORT**  
 1323                   **ELSE NO\_OPTION**  
 1324                   Conformance for *sem\_trywait*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

## 1325    **11.2.7    Unlock a Semaphore**

1326    Function: *sem\_post()*

### 1327    **11.2.7.1 Synopsis**

1328    **1**  
 1329                   *M\_GA\_stdC\_proto\_decl(int; sem\_post; sem\_t \*sem, semaphore.h;;)*  
 1330                   **SEE:**    Assertion *GA\_stdC\_proto\_decl* in §2.7.3  
 1331                   Conformance for *sem\_post*: *PASS[1, 2], NO\_OPTION*

1332    **2**  
 1333                   *M\_GA\_commonC\_int\_result\_decl(sem\_post; semaphore.h;;)*  
 1334                   **SEE:**    Assertion *GA\_commonC\_int\_result\_decl* in §2.7.3  
 1335                   Conformance for *sem\_post*: *PASS[1, 2], NO\_OPTION*

1336    **3**  
 1337                   *M\_GA\_macro\_result\_decl(int; sem\_post; semaphore.h;;)*  
 1338                   **SEE:**    Assertion *GA\_macro\_result\_decl* in §1.3.4  
 1339                   Conformance for *sem\_post*: *PASS, NO\_OPTION*

1340    **4**  
 1341                   *M\_GA\_macro\_args ( sem\_post; semaphore.h;;)*  
 1342                   **SEE:**    Assertion *GA\_macro\_args* in §2.7.3  
 1343                   Conformance for *sem\_post*: *PASS, NO\_OPTION*

### 1344    **11.2.7.2 Description**

1345    **sem\_post**  
 1346                   **FOR:**    *sem\_init()* and *sem\_open()*  
 1347                   **IF** *PCTS\_sem\_post* **THEN**  
 1348                    **IF** *PCTS\_function* **THEN**  
 1349                      **SETUP:**   Create a semaphore using *function()*.  
 1350                      **TEST:**    A successful call to *sem\_post()*, unlocks the semaphore referenced by *sem*  
 1351   by performing the semaphore unlock operation on that semaphore, and returns  
 1352   the value zero.  
 1353                      **TR:**    When testing for *sem\_init()*, perform the test consistent with the flag  
 1354   *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1355   not a way to get appropriate privilege to call *sem\_init()*.  
 1356                      **NOTE:**   The assertion is tested once for each function specified in the FOR clause. The  
 1357   assertion is to be read by substituting *function()* with the current function  
 1358   specified in the FOR clause. The name of the function also is to be substituted  
 1359   for each occurrence in the construct *PCTS\_function*.  
 1360                   **ELSE NO\_TEST\_SUPPORT**

1361           **ELSE NO\_OPTION**  
1362           *Conformance for sem\_post: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1363       **5**       **FOR:**     *sem\_init()* and *sem\_open()*  
1364           **IF PCTS\_sem\_post THEN**  
1365               **IF PCTS\_function THEN**  
1366                   **SETUP:**   Create a semaphore using *function()*.  
1367                   **TEST:**     When the semaphore value resulting from *sem\_post()* is positive, then the  
1368                               semaphore value is incremented.  
1369                   **ELSE NO\_TEST\_SUPPORT**  
1370           **ELSE NO\_OPTION**  
1371           *Conformance for sem\_post: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1372       **6**       **FOR:**     *sem\_init()* and *sem\_open()*  
1373           **IF PCTS\_sem\_post THEN**  
1374               **IF PCTS\_sem\_wait THEN**  
1375                   **SETUP:**   Create a semaphore using *function()*. Also, create multiple processes and have  
1376                               them block waiting on the semaphore.  
1377                   **TEST:**     When the value of the semaphore resulting from *sem\_post()* is zero, then one  
1378                               of the processes blocked waiting for the semaphore returns successfully from  
1379                               its call to *sem\_wait()*.  
1380                   **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1381                               *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1382                               not a way to get appropriate privilege to call *sem\_init()*.  
1383                   **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
1384                               assertion is to be read by substituting *function()* with the current function  
1385                               specified in the FOR clause. The name of the function also is to be substituted  
1386                               for each occurrence in the construct *PCTS\_function*.  
1387                   **ELSE NO\_TEST\_SUPPORT**  
1388           **ELSE NO\_OPTION**  
1389           *Conformance for sem\_post: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1390       **R\_1 FOR:**   *sem\_init()* and *sem\_open()*  
1391           **IF PCTS\_sem\_post THEN**  
1392               **IF PCTS\_function and {\_POSIX\_PRIORITY\_SCHEDULING} THEN**  
1393                   **SETUP:**   Create a semaphore using *function()*.  
1394                   **TEST:**     When the value of the semaphore resulting from *sem\_post()* is zero, the process  
1395                               to be unblocked is chosen in a manner appropriate to the scheduling policies  
1396                               and parameters in effect for the blocked processes.  
1397                   **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1398                               *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1399                               not a way to get appropriate privilege to call *sem\_init()*.  
1400                   **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
1401                               assertion is to be read by substituting *function()* with the current function  
1402                               specified in the FOR clause. The name of the function also is to be substituted  
1403                               for each occurrence in the construct *PCTS\_function*.  
1404                   **ELSE NO\_TEST\_SUPPORT**  
1405           **ELSE NO\_OPTION**  
1406           **SEE:**     Assertion 8 in §11.2.7.2.

1407       **7**       **FOR:**     *sem\_init()* and *sem\_open()*  
1408           **IF PCTS\_sem\_post THEN**  
1409               **IF PCTS\_function and {\_POSIX\_PRIORITY\_SCHEDULING} THEN**  
1410                   **SETUP:**   Create a semaphore using *function()*.  
1411                   **TEST:**     When the value of the semaphore resulting from *sem\_post()* is zero, and the  
1412                               scheduler is *SCHED\_FIFO* or *SCHED\_RR*, the highest priority waiting process is  
1413                               unblocked.

1414 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 1415 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1416 not a way to get appropriate privilege to call *sem\_init()*.

1417 Test for each of *SCHED\_FIFO* and *SCHED\_RR*.  
 1418 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
 1419 assertion is to be read by substituting *function()* with the current function  
 1420 specified in the FOR clause. The name of the function also is to be substituted  
 1421 for each occurrence in the construct *PCTS\_function*.

1422 **ELSE NO\_TEST\_SUPPORT**  
 1423 **ELSE NO\_OPTION**  
 1424 *Conformance for sem\_post: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1425 **8 FOR:** *sem\_init()* and *sem\_open()*  
 1426 **IF** *PCTS\_sem\_post* **THEN**  
 1427 **IF** *PCTS\_function* and *{\_POSIX\_PRIORITY\_SCHEDULING}* **THEN**  
 1428 **SETUP:** Create a semaphore using *function()*.  
 1429 **TEST:** When the value of the semaphore resulting from *sem\_post()* is zero, and the  
 1430 scheduler is *SCHED\_FIFO* or *SCHED\_RR*, and there is more than one highest  
 1431 priority process blocked waiting for the semaphore, then the highest priority  
 1432 process that has been waiting the longest is unblocked.  
 1433 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 1434 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1435 not a way to get appropriate privilege to call *sem\_init()*.

1436 Test for each of *SCHED\_FIFO* and *SCHED\_RR*.  
 1437 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
 1438 assertion is to be read by substituting *function()* with the current function  
 1439 specified in the FOR clause. The name of the function also is to be substituted  
 1440 for each occurrence in the construct *PCTS\_function*.

1441 **ELSE NO\_TEST\_SUPPORT**  
 1442 **ELSE NO\_OPTION**  
 1443 *Conformance for sem\_post: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1444 **D\_1 IF** *PCTS\_sem\_post* and a PCD.1b documents the following **THEN**  
 1445 **TEST:** A PCD.1b that documents the choice of a process to unblock if  
 1446 *{\_POSIX\_PRIORITY\_SCHEDULING}* is defined does so in §11.2.7.2.  
 1447 **ELSE NO\_OPTION**  
 1448 *Conformance for sem\_post: PASS, NO\_OPTION*

1449 **9 FOR:** *sem\_init()* and *sem\_open()*  
 1450 **IF** *PCTS\_sem\_post* **THEN**  
 1451 **IF** *PCTS\_function* **THEN**  
 1452 **SETUP:** Create a semaphore using *function()*.  
 1453 **TEST:** The *sem\_post()* function is reentrant with respect to signals and may be  
 1454 invoked from a signal-catching function.  
 1455 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 1456 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1457 not a way to get appropriate privilege to call *sem\_init()*.  
 1458 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
 1459 assertion is to be read by substituting *function()* with the current function  
 1460 specified in the FOR clause. The name of the function also is to be substituted  
 1461 for each occurrence in the construct *PCTS\_function*.

1462 **ELSE NO\_TEST\_SUPPORT**  
 1463 **ELSE NO\_OPTION**  
 1464 *Conformance for sem\_post: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1465 **D\_2 IF** *PCTS\_sem\_post* and a PCD.1b documents the following **THEN**

1466                   **TEST:**     A PCD.1b that documents whether or not it supports the *sem\_post()* function does so  
 1467   in §11.2.7.2.  
 1468                   **ELSE NO\_OPTION**  
 1469                   *Conformance for sem\_post: PASS, NO\_OPTION*

### 1470    11.2.7.3 Returns

1471    **R\_2 FOR:**     *sem\_init()* and *sem\_open()*  
 1472                   **IF** *PCTS\_sem\_post* **THEN**  
 1473                    **IF** *PCTS\_function* **THEN**  
 1474                      **SETUP:**    Create a semaphore using *function()*.  
 1475                      **TEST:**     When a call to *sem\_post()* completes successfully, the interface returns a value  
 1476   of 0.  
 1477                      **TR:**     When testing for *sem\_init()*, perform the test consistent with the flag  
 1478   *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1479   not a way to get appropriate privilege to call *sem\_init()*.  
 1480                      **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
 1481   assertion is to be read by substituting *function()* with the current function  
 1482   specified in the FOR clause. The name of the function also is to be substituted  
 1483   for each occurrence in the construct *PCTS\_function*.  
 1484                    **ELSE NO\_TEST\_SUPPORT**  
 1485                    **ELSE NO\_OPTION**  
 1486                    **SEE:**     Assertion *sem\_post* in §11.2.7.2

1487    **R\_3 FOR:**     *sem\_init()* and *sem\_open()*  
 1488                   **IF** *PCTS\_sem\_post* **THEN**  
 1489                    **IF** *PCTS\_function* **THEN**  
 1490                      **SETUP:**    Create a semaphore using *function()*.  
 1491                      **TEST:**     When a call to *sem\_post()* completes unsuccessfully, the interface returns a  
 1492   value of -1 and sets *errno* to indicate the error.  
 1493                      **TR:**     When testing for *sem\_init()*, perform the test consistent with the flag  
 1494   *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1495   not a way to get appropriate privilege to call *sem\_init()*.  
 1496                      **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
 1497   assertion is to be read by substituting *function()* with the current function  
 1498   specified in the FOR clause. The name of the function also is to be substituted  
 1499   for each occurrence in the construct *PCTS\_function*.  
 1500                    **ELSE NO\_TEST\_SUPPORT**  
 1501                    **ELSE NO\_OPTION**  
 1502                    **SEE:**     All assertions in §11.2.7.4

### 1503    11.2.7.4 Errors

1504    **10    FOR:**     *sem\_init()* and *sem\_open()*  
 1505                   **IF** *PCTS\_sem\_post* **THEN**  
 1506                    **IF** *PCTS\_function* **THEN**  
 1507                      **SETUP:**    Create a semaphore using *function()*.  
 1508                      **TEST:**     A call to *sem\_post()*, when the *sem* does not refer to a valid semaphore returns  
 1509   a value of -1 and sets *errno* to [EINVAL].  
 1510                      **TR:**     When testing for *sem\_init()*, perform the test consistent with the flag  
 1511   *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1512   not a way to get appropriate privilege to call *sem\_init()*.  
 1513                      **NOTE:**    A subroutine is recommended that either returns an invalid semaphore or  
 1514   indicates that there is no way to generate an invalid semaphore on the system.

1515   The assertion is tested once for each function specified in the FOR clause. The  
 1516   assertion is to be read by substituting *function()* with the current function

1517 specified in the FOR clause. The name of the function also is to be substituted  
 1518 for each occurrence in the construct *PCTS\_function*.  
 1519 **ELSE NO\_TEST\_SUPPORT**  
 1520 **ELSE NO\_OPTION**  
 1521 *Conformance for sem\_post: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1522 **11 IF** not *PCTS\_sem\_post* **THEN**  
 1523 **TEST:** A call to *sem\_post()* returns a value of -1 and sets *errno* to [ENOSYS].  
 1524 **ELSE NO\_OPTION**  
 1525 *Conformance for sem\_post: PASS, NO\_OPTION*

## 1526 **11.2.8 Get the Value of a Semaphore**

1527 Function: *sem\_getvalue()*

### 1528 **11.2.8.1 Synopsis**

1529 **1**  
 1530 *M\_GA\_stdC\_proto\_decl(int; sem\_getvalue; sem\_t \*sem, int \*sval; semaphore.h;;)*  
 1531 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 1532 *Conformance for sem\_getvalue: PASS[1, 2], NO\_OPTION*

1533 **2**  
 1534 *M\_GA\_commonC\_int\_result\_decl(sem\_getvalue; semaphore.h;;)*  
 1535 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 1536 *Conformance for sem\_getvalue: PASS[1, 2], NO\_OPTION*

1537 **3**  
 1538 *M\_GA\_macro\_result\_decl(int; sem\_getvalue; semaphore.h;;)*  
 1539 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 1540 *Conformance for sem\_getvalue: PASS, NO\_OPTION*

1541 **4**  
 1542 *M\_GA\_macro\_args ( sem\_getvalue; semaphore.h;;)*  
 1543 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 1544 *Conformance for sem\_getvalue: PASS, NO\_OPTION*

### 1545 **11.2.8.2 Description**

#### 1546 **sem\_getvalue**

1547 **FOR:** *sem\_init()* and *sem\_open()*  
 1548 **IF** *PCTS\_sem\_getvalue* **THEN**  
 1549 **IF** *PCTS\_function* **THEN**  
 1550 **SETUP:** Create a semaphore using *function()*.  
 1551 **TEST:** A successful call of *sem\_getvalue(sem, sval)* updates the location referenced  
 1552 by the *sval* argument to have the value of the semaphore referenced by *sem* at  
 1553 some unspecified time during the call, and returns the value to zero.  
 1554 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 1555 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1556 not a way to get appropriate privilege to call *sem\_init()*.  
 1557 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
 1558 assertion is to be read by substituting *function()* with the current function  
 1559 specified in the FOR clause. The name of the function also is to be substituted  
 1560 for each occurrence in the construct *PCTS\_function*.  
 1561 **ELSE NO\_TEST\_SUPPORT**  
 1562 **ELSE NO\_OPTION**  
 1563 *Conformance for sem\_getvalue: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*



1564 **5**       **FOR:**     *sem\_init()* and *sem\_open()*  
 1565       **IF** *PCTS\_sem\_getvalue* **THEN**  
 1566           **IF** *PCTS\_function* **THEN**  
 1567               **SETUP:** Create a semaphore using *function()*.  
 1568               **TEST:**    The *sem\_getvalue()* function does not affect the state of the semaphore  
 1569               referenced by *sem*.  
 1570               **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 1571               *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1572               not a way to get appropriate privilege to call *sem\_init()*.  
 1573               **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
 1574               assertion is to be read by substituting *function()* with the current function  
 1575               specified in the FOR clause. The name of the function also is to be substituted  
 1576               for each occurrence in the construct *PCTS\_function*.  
 1577               **ELSE** *NO\_TEST\_SUPPORT*  
 1578       **ELSE** *NO\_OPTION*  
 1579       Conformance for *sem\_getvalue*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1580 **6**       **FOR:**     *sem\_init()* and *sem\_open()*  
 1581       **IF** *PCTS\_sem\_getvalue* **THEN**  
 1582           **IF** *PCTS\_function* **THEN**  
 1583               **SETUP:** Create a semaphore using *function()*.  
 1584               **TEST:**    When *sem* is locked, then the value returned by *sem\_getvalue()* is either zero  
 1585               or a negative number whose absolute value represents the number of processes  
 1586               waiting for the semaphore at some unspecified time during the call.  
 1587               **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 1588               *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1589               not a way to get appropriate privilege to call *sem\_init()*.  
 1590               **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
 1591               assertion is to be read by substituting *function()* with the current function  
 1592               specified in the FOR clause. The name of the function also is to be substituted  
 1593               for each occurrence in the construct *PCTS\_function*.  
 1594               **ELSE** *NO\_TEST\_SUPPORT*  
 1595       **ELSE** *NO\_OPTION*  
 1596       Conformance for *sem\_getvalue*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1597 **D\_1** **IF** *PCTS\_sem\_getvalue* and a PCD.1b documents the following **THEN**  
 1598       **TEST:**    A PCD.1b that documents whether or not it supports the *sem\_getvalue()* function does  
 1599       so in §11.2.8.2.  
 1600       **ELSE** *NO\_OPTION*  
 1601       Conformance for *sem\_getvalue*: *PASS, NO\_OPTION*

### 1602 11.2.8.3 Returns

1603 **R\_1** **FOR:**     *sem\_init()* and *sem\_open()*  
 1604       **IF** *PCTS\_sem\_getvalue* **THEN**  
 1605           **IF** *PCTS\_function* **THEN**  
 1606               **SETUP:** Create a semaphore using *function()*.  
 1607               **TEST:**    When a call to *sem\_getvalue()* completes successfully, the interface returns a  
 1608               value of 0.  
 1609               **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
 1610               *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
 1611               not a way to get appropriate privilege to call *sem\_init()*.  
 1612               **NOTE:**    The assertion is tested once for each function specified in the FOR clause. The  
 1613               assertion is to be read by substituting *function()* with the current function  
 1614               specified in the FOR clause. The name of the function also is to be substituted  
 1615               for each occurrence in the construct *PCTS\_function*.  
 1616               **ELSE** *NO\_TEST\_SUPPORT*  
 1617       **ELSE** *NO\_OPTION*  
 1618       **SEE:**     Assertion *sem\_getvalue* in §11.2.8.2

1619 **R\_2 FOR:** *sem\_init()* and *sem\_open()*  
1620 **IF** *PCTS\_sem\_getvalue* **THEN**  
1621 **IF** *PCTS\_function* **THEN**  
1622 **SETUP:** Create a semaphore using *function()*.  
1623 **TEST:** When a call to *sem\_getvalue()* completes unsuccessfully, the interface returns  
1624 a value of -1 and sets *errno* to indicate the error.  
1625 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1626 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1627 not a way to get appropriate privilege to call *sem\_init()*.  
1628 **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
1629 assertion is to be read by substituting *function()* with the current function  
1630 specified in the FOR clause. The name of the function also is to be substituted  
1631 for each occurrence in the construct *PCTS\_function*.  
1632 **ELSE NO\_TEST\_SUPPORT**  
1633 **ELSE NO\_OPTION**  
1634 **SEE:** All assertions in §11.2.8.4

1635 **7 FOR:** *sem\_init()* and *sem\_open()*  
1636 **IF** *PCTS\_sem\_getvalue* **THEN**  
1637 **IF** *PCTS\_function* and *PCTS\_SEM\_INVALID* **THEN**  
1638 **SETUP:** Create a semaphore using *function()*.  
1639 **TEST:** A call to *sem\_getvalue()*, when the *sem* argument does not refer to a valid  
1640 semaphore, returns a value of -1 and sets *errno* to [EINVAL]  
1641 **TR:** When testing for *sem\_init()*, perform the test consistent with the flag  
1642 *PCTS\_GAP\_sem\_init*; that is, generate a *NO\_TEST\_SUPPORT* test result code if there is  
1643 not a way to get appropriate privilege to call *sem\_init()*.  
1644 **NOTE:** A subroutine is recommended that either returns an invalid semaphore or  
1645 indicates that there is no way to generate an invalid semaphore on the system.  
  
1646 The assertion is tested once for each function specified in the FOR clause. The  
1647 assertion is to be read by substituting *function()* with the current function  
1648 specified in the FOR clause. The name of the function also is to be substituted  
1649 for each occurrence in the construct *PCTS\_function*.  
1650 **ELSE NO\_TEST\_SUPPORT**  
1651 **ELSE NO\_OPTION**  
1652 *Conformance for sem\_getvalue: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1653 **8 IF** not *PCTS\_sem\_getvalue* **THEN**  
1654 **TEST:** A call to *sem\_getvalue()* returns a value of -1 and sets *errno* to [ENOSYS].  
1655 **ELSE NO\_OPTION**  
1656 *Conformance for sem\_getvalue: PASS, NO\_OPTION*

This page is intentionally blank.

## Section 12: Memory Management

- 180 **1 TEST:** The page size, in bytes, is the value of the configurable system variable [PAGESIZE].  
 181 *Conformance for mem\_intro: PASS*
- 182 **D\_1 IF** a PCD.1b documents the following **THEN**  
 183 **TEST:** A PCD.1b that documents range lockings and mappings are restricted  
 184 o page-size boundaries, does so in § 12.  
 185 **ELSE NO\_OPTION**  
 186 *Conformance for mem\_intro: PASS, NO\_OPTION*
- 187 **D\_2 IF** a PCD.1b documents the following **THEN**  
 188 **TEST:** A PCD.1b that documents 1B page size, meaning no restrictions on size  
 189 or alignment, of range lockings and mappings, does so in § 12.  
 190 **ELSE NO\_OPTION**  
 191 *Conformance for mem\_intro: PASS, NO\_OPTION*
- 192 **D\_3 IF** a PCD.1b documents the following **THEN**  
 193 **TEST:** A PCD.1b that documents whether locking memory guarantees fixed  
 194 translation between virtual addresses (as seen by the process) and  
 195 physical addresses, does so in § 12.  
 196 **ELSE NO\_OPTION**  
 197 *Conformance for mem\_intro: PASS, NO\_OPTION*
- 198 **R\_1 IF** *PCTS\_mlockall* or *PCTS\_mlock* **THEN**  
 199 **TEST:** Per-process memory locks are not inherited across a *fork()*.  
 200 **ELSE NO\_TEST\_SUPPORT**  
 201 **SEE:** Assertion 2 in §3.1.1.2
- 202 **R\_2 IF** *PCTS\_mlockall* or *PCTS\_mlock* **THEN**  
 203 **TEST:** All memory locks owned by a process are unlocked upon *exec* or  
 204 process termination.  
 205 **ELSE NO\_TEST\_SUPPORT**  
 206 **SEE:** Assertion mlock in §12.1.2.2
- 207 **R\_3 IF** *PCTS\_munlock* **THEN**  
 208 **TEST:** Unmapping of an address range remove any memory locks  
 209 established on that address range by this process.  
 210 **ELSE NO\_TEST\_SUPPORT**  
 211 **SEE:** Assertion munlock\_remove\_maps in §12.2.2.2
- 212 **2 IF** *PCTS\_mmap* **THEN**

213                   **TEST:** Once a file is "mapped" into a process address space, the data can be  
 214                   manipulated as memory.  
 215                   **ELSE NO\_TEST\_SUPPORT**  
 216                   *Conformance for mem\_intro: PASS, NO\_TEST\_SUPPORT*

217   **3**           **IF PCTS\_mmap THEN**  
 218                   **TEST:** When more than one process maps a file, its contents are shared among  
 219                   them..  
 220                   **ELSE NO\_TEST\_SUPPORT**  
 221                   *Conformance for mem\_intro: PASS, NO\_TEST\_SUPPORT*

222   **4**           **IF PCTS\_mmap THEN**  
 223                   **TEST:** When the mappings allow shared write access, then data written into  
 224                   the memory object through the address space of one process appear in  
 225                   the address spaces of all processes that similarly map the same portion  
 226                   of the memory object.  
 227                   **ELSE NO\_TEST\_SUPPORT**  
 228                   *Conformance for mem\_intro: PASS, NO\_TEST\_SUPPORT*

229   **R\_4**       **IF PCTS\_shm\_unlink THEN**  
 230                   **TEST:** *unlink ( )* of a mapped file or *shm-unlink ( )* of a shared memory object,  
 231                   while causing the removal of the name, does not unmap any mappings  
 232                   while causing the removal of the name, does not unmap any mappings  
 233                   established for the object. Once the name has been removed, the  
 234                   contents of the memory object are preserved as long as a process has  
 235                   the memory object open or has some area of the memory object  
 236                   mapped.  
 237                   **ELSE NO\_TEST\_SUPPORT**  
 238                   **SEE:** All assertions in §12.3.2.2

239   **R\_5**       **IF {\_POSIX\_MEMORY\_PROTECTION} THEN**  
 240                   **IF PCTS\_mmap THEN**  
 241                   **TEST:** References to whole pages within the mapping but beyond the  
 242                   current length of an object result in a SIGBUS signal.  
 243                   **ELSE NO\_TEST\_SUPPORT**  
 244                   **ELSE NO\_OPTION**  
 245                   **SEE:** Assertion *mmap\_ SIGBUS* in §12.1.2.2

246   **D\_4**       **IF {\_POSIX\_MEMORY\_PROTECTION} and a PCD.1b documents the following THEN**  
 247                   **TEST:** A PCD.1b that documents the result of references to memory within the  
 248                   mapping but beyond the current length of an object, does so in §12.  
 249                   **ELSE NO\_OPTION**  
 250                   *Conformance for mem\_intro: PASS, NO\_OPTION*

251   **5**           **IF {\_POSIX\_MEMORY\_PROTECTION} THEN**  
 252                   **IF: PCTS\_mmap THEN**  
 253                   **SETUP:** Create a mapped memory object using *mmap ( )*.  
 254                   **TEST:** The size of a memory object is unaffected by access beyond the end of  
 255                   the object.  
 256                   **ELSE NO\_TEST\_SUPPORT**  
 257                   **ELSE NO\_OPTION**

258 *Conformance for mem\_intro: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*  
 259 **6** **IF** {\_POSIX\_MEMORY\_PROTECTION} **THEN**  
 260 **IF:** *PCTS\_mmap* **THEN**  
 261 **SETUP:** Create a mapped memory object using without write access using *mmap* (.  
 262 **TEST:** Write attempts to memory that was mapped without write access, results  
 263 in a SIGSEGV signal.  
 264 **ELSE** *NO\_TEST\_SUPPORT*  
 265 **ELSE** *NO\_OPTION*  
 266 *Conformance for mem\_intro: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

267 **7** **IF** {\_POSIX\_MEMORY\_PROTECTION} **THEN**  
 268 **IF:** *PCTS\_mmap* **THEN**  
 269 **SETUP:** Create a mapped memory object using with PROT\_NONE using *mmap* (.  
 270 **TEST:** Any access to memory mapped PROT\_NONE , results in a SIGSEGV  
 271 signal.  
 272 **ELSE** *NO\_TEST\_SUPPORT*  
 273 **ELSE** *NO\_OPTION*  
 274 *Conformance for mem\_intro: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

275 **R\_6** **IF** {\_POSIX\_MEMORY\_PROTECTION} **THEN**  
 276 **TEST:** References to unmapped addresses result in a SIGSEGV signal.  
 277 **ELSE** *NO\_OPTION*  
 278 **SEE:** Assertion munmap\_ SIGSEV in §12.2.2.2

279 **D\_5** **IF** not {\_POSIX\_MEMORY\_PROTECTION} and a PCD.1b documents the following **THEN**  
 280 **TEST:** A PCD.1b that documents the effect of references to unmapped addresses, does so in  
 281 §12.  
 282 **ELSE** *NO\_OPTION*  
 283 *Conformance for mem\_intro: PASS, NO\_OPTION*

## 284 12.1 Memory Locking Functions

### 285 12.1.1 Lock/Unlock the Address Space of a Process

286 Functions: *mlockall()*, *munlockall()*

#### 287 12.1.1.1 Synopsis

288 **1**

289 *M\_GA\_stdC\_proto\_decl(int; mlockall; int; int flags; sys/mman .h;;;)*

290 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3

291 *Conformance for mlockall: PASS[1, 2], NO\_OPTION*

292 **2**

293 *M\_GA\_commonC\_int\_result\_decl(mlockall; sys/mman.h;;;)*

294 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3

295 *Conformance for mlockall: PASS[1, 2], NO\_OPTION*

296 **3**

297 *M\_GA\_macro\_result\_decl(int; mlockall; sys/mman.h;;;)*

298 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4

299 *Conformance for mlockall: PASS, NO\_OPTION*

300 **4**

301 *M\_GA\_macro\_args ( mlockall; sys/mman.h;;;)*

302 **SEE:** Assertion GA\_macro\_args in §2.7.3

303 *Conformance for mlockall: PASS, NO\_OPTION*

304 **5**

305 *M\_GA\_stdC\_proto\_decl(int; munlockall; sys/mman .h;;;)*

306 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3

307 *Conformance for munlockall: PASS[5, 6], NO\_OPTION*

308 **6**

309 *M\_GA\_commonC\_int\_result\_decl(munlockall; sys/mman.h;;;)*

310 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3

311 *Conformance for munlockall: PASS[5, 6], NO\_OPTION*

312 **7**

313 *M\_GA\_macro\_result\_decl(int; munlockall; sys/mman.h;;;)*

314 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4

315 *Conformance for munlockall: PASS, NO\_OPTION*

316 **8**

317 *M\_GA\_macro\_args ( munlockall; sys/mman.h;;;)*

318 **SEE:** Assertion GA\_macro\_args in §2.7.3

319 *Conformance for munlockall: PASS, NO\_OPTION*

#### 320 12.1.1.2 Description

```

321     mlockall
322     FOR: execl(), execv(), execle(), execve(), execl(), and execve()
323     IF PCTS_mlockall THEN
324         IF: PCTS_GAP_mlockall THEN
325             TEST: A successful call to the function mlockall() returns zero and makes all of the pages
326 mapped by the address space of a process memory resident until unlocked, or until
327 the process exits or executes a successful call to function().
328             NOTE: The assertion is tested once for each function specified in the FOR clause. The
329 assertion is to be read by substituting function() with the current function specified
330 in the FOR clause. The name of the function also is to be substituted for each
331 occurrence in the construct PCTS_function
332             TR: Try both exec and exit().
333         ELSE NO_TEST_SUPPORT
334     ELSE NO_OPTION
335     Conformance for sem_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

336 9     IF PCTS_mlockall THEN
337         SETUP: Include the header <sys/mman.h>
338         TEST: The constants MCL_CURRENT and MCL_FUTURE are defined and are bitwise
339 distinct.
340     ELSE NO_OPTION
341     Conformance for mlockall: PASS, NO_OPTION

342 10    IF PCTS_mlockall THEN
343         IF: PCTS_GAP_mlockall THEN
344             TEST: When the flag MCL_CURRENT is set, the call mlockall(flags) locks all of the
345 pages currently mapped into the address space of the process.
346             TR: Test for at least two disjoint sets of pages.
347         ELSE NO_TEST_SUPPORT
348     ELSE NO_OPTION
349     Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

350 10    IF PCTS_mlockall THEN
351         IF: PCTS_GAP_mlockall THEN
352             TEST: When the flag MCL_FUTURE is set, the call mlockall(flags) locks all of the
353 pages that become mapped into the address space of the process in the future,
354 when those mappings are established.
355             TR: Test for at least two disjoint sets of pages.
356         ELSE NO_TEST_SUPPORT
357     ELSE NO_OPTION
358     Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

359 D_1   IF PCTS_mlockall THEN
360         TEST: The PCD.1b that documents the behavior if MCL_FUTURE is specified, and
361 any of
362         7. the automatic locking of future mappings eventually causes the
363 amounts of locked memory to exceed the amount of available
364 physical memory
365         8. the automatic locking of future mappings eventually causes the
366 amount of locked memory to exceed any other implementation-
367 defined limit,
368         9. the manner in which the implementation informs the application
369 of these situations
370         in §12.1.1.2.
371     ELSE NO_OPTION
372     Conformance for mlockall: PASS, NO_OPTION

```



373 **munlockall**  
374 **IF** *PCTS\_munlockall* **THEN**  
375 **TEST:** A successful call to *munlockall*() unlocks all currently mapped pages of the address  
376 space of the process, with respect to the process's address space, and returns zero.  
377 **TR:** Test for at least two disjoint sets of pages.  
378 **ELSE** *NO\_TEST\_SUPPORT*  
379 **ELSE** *NO\_OPTION*  
380 *Conformance for munlockall: PASS, NO\_OPTION*

381 **12** **IF** *PCTS\_munlockall* **THEN**  
382 **IF:** *PCTS\_GAP\_mlockall* **THEN**  
383 **TEST:** Any pages that become mapped into the address space of the process after a  
384 call to *munlockall*() are not locked, unless there is an intervening call to  
385 *mlockall*() specifying *MCL\_FUTURE* or a subsequent call to *mlockall*()  
386 specifying *MCL\_CURRENT*.  
387 **TR:** Test for both *MCL\_FUTURE* and *MCL\_CURRENT*.  
388 **ELSE** *NO\_TEST\_SUPPORT*  
389 **ELSE** *NO\_OPTION*  
390 *Conformance for mlockall: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

391 **13** **IF** *PCTS\_munlockall* **THEN**  
392 **TEST:** When pages mapped into the address space of the process are also mapped into the  
393 address spaces of other processes, and are locked by those processes, the locks  
394 established by the other processes are unaffected by a call by this process to  
395 *munlockall*().  
396 **TR:** Test for at least two other processes.  
397 **ELSE** *NO\_OPTION*  
398 *Conformance for mlockall: PASS, NO\_OPTION*

399 **14** **IF** *PCTS\_mlockall* **THEN**  
400 **IF:** *PCTS\_GAP\_munlockall* **THEN**  
401 **TEST:** After a successful call to *mlockall*( ) that specifies *MCL\_CURRENT*, all  
402 currently mapped pages of the process's address space are memory resident and  
403 locked.  
404 **ELSE** *NO\_TEST\_SUPPORT*  
405 **ELSE** *NO\_OPTION*  
406 *Conformance for mlockall: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

407 **D\_2** **IF** *PCTS\_mlockall* and a PCD.1b documents the following **THEN**  
408 **TEST:** A PCD.1b that documents the memory residency of unlocked pages  
409 does so in §12.1.1.2.  
410 **ELSE** *NO\_OPTION*  
411 *Conformance for mlockall: PASS, NO\_OPTION*

412 **15** **IF** *PCTS\_mlockall* **THEN**  
413 **IF:** *PCTS\_RAP\_mlockall* **THEN**  
414 **TEST:** Appropriate privilege is required to lock process memory with *mlockall*( ).  
415 **ELSE** *NO\_TEST\_SUPPORT*  
416 **ELSE** *NO\_OPTION*  
417 *Conformance for mlockall: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

418 **D\_3** **IF** *PCTS\_mlockall* and a PCD.1b documents the following **THEN**  
419 **TEST:** A PCD.1b that documents whether or not it supports the *mlockall*( )  
420 function does so in §12.1.1.2.  
421 **ELSE** *NO\_OPTION*  
422 *Conformance for mlockall: PASS, NO\_OPTION*

423 **D\_4** **IF** *PCTS\_munlockall* and a PCD.1b documents the following **THEN**

424                   **TEST:** A PCD.1b that documents whether or not it supports the *munlockall()*  
425                   function does so in §12.1.1.2.  
426                   **ELSE NO\_OPTION**  
427                   *Conformance for mlockall: PASS, NO\_OPTION*

428   **12.1.1.3 Returns**

429   **R\_1 IF PCTS\_mlockall THEN**  
430                   **TEST:** When a call to *mlockall()* completes successfully, the interface returns  
431                   a value of 0.  
432                   **ELSE NO\_TEST\_SUPPORT**  
433                   **ELSE NO\_OPTION**  
434                   **SEE:** Assertion mlockall in §12.1.1.2

435   **R\_2 IF PCTS\_mlockall THEN**  
436                   **TEST:** When a call to *mlockall()* completes unsuccessfully, the interface  
437                   returns a value of -1, sets *errno* to indicate the error, and no additional  
438                   memory is locked.  
439                   **ELSE NO\_OPTION**  
440                   **SEE:** All assertions in §12.1.1.4 controlled by a *PCTS\_mlockall*

441   **D\_5 IF PCTS\_mlockall and a PCD.1b documents the following THEN**  
442                   **TEST:** A PCD.1b that documents the effect of failure of *mlockall()* on  
443                   previously existing locks in the address space does so in §12.1.1.3.  
444

445   **R\_3 IF PCTS\_mlockall and a PCD.1b documents the following PCTS\_munlockall THEN**  
446                   **TEST:** The interface *munlockall()* returns a value of 0  
447                   **ELSE NO\_OPTION**  
448                   **SEE:** Assertion munlockall in §12.1.1.2

449   **12.1.1.4 Errors**

450   **16 IF not PCTS\_mlockall THEN:**  
451                   **TEST:** A call to *mlockall()*, returns a value of -1 and sets *errno* to [ENOSYS].  
452                   **ELSE NO\_OPTION**  
453                   *Conformance for mlockall: PASS, NO\_OPTION*

454   **17 IF not PCTS\_munlockall THEN:**  
455                   **TEST:** A call to *munlockall()*, returns a value of -1 and sets *errno* to [ENOSYS].  
456                   **ELSE NO\_OPTION**  
457                   *Conformance for munlockall: PASS, NO\_OPTION*

458   **18 IF PCTS\_mlockall THEN:**  
459                   **IF PCTS\_GAP\_mlockall THEN:**  
460                   **TEST:** A call to *mlockall()*, when some or all of the memory identified by the operation  
461                   could not be locked, returns a value of -1 and sets *errno* to [EAGAIN].  
462                   **ELSE NO\_TEST\_SUPPORT**  
463                   **ELSE NO\_OPTION**  
464                   *Conformance for mlockall: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

465   **19 IF PCTS\_mlockall THEN:**  
466                   **IF PCTS\_GAP\_mlockall THEN:**  
467                   **TEST:** A call to *mlockall()*, when the *flags* argument is zero, returns a value of -1 and  
468                   sets *errno* to [EINVAL].

469                   **ELSE NO\_TEST\_SUPPORT**  
470                   **ELSE NO\_OPTION**  
471                   *Conformance for mlockall: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

472   **20**           **IF PCTS\_mlockall THEN**  
473                   **IF: PCTS\_GAP\_mlockall THEN:**  
474                   **TEST:**    A call to *mlockall()*, when the *flags* includes unimplemented flags, returns a  
475                                    value of -1 and sets *errno* to [EINVAL].  
476                   **NOTE:**    A subroutine is recommended that either returns a *flags* argument that includes  
477                                    unimplemented flags or indicates that there is no way to generate a *flags*  
478                                    argument that includes unimplemented flags on the system.  
479                   **ELSE NO\_TEST\_SUPPORT**  
480                   **ELSE NO\_OPTION**  
481                   *Conformance for mlockall: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

482   **21**           **IF PCTS\_mlockall THEN**  
483                   **IF: PCTS\_GAP\_mlockall**  
484                   **PCTS\_DETECT\_LOCKABLE\_MEMORY\_LIMITS\_mlockall THEN:**  
485                   **TEST:**    A call to *mlockall()*, when locking all of the pages currently mapped into the  
486                                    address space of the process would exceed an implementation-defined limit on  
487                                    the amount of memory that the process may lock, , returns a value of -1 and  
488                                    sets *errno* to [ENOMEM].  
489                   **ELSE NO\_TEST\_SUPPORT**  
490                   **ELSE NO\_OPTION**  
491                   *Conformance for mlockall: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

492   **D\_6**           **IF PCTS\_mlockall THEN**  
493                   **TEST:**    The PCD.1b documents the maximum amount of memory that process  
494                                    may lock in §12.1.1.4.  
495                   **ELSE NO\_OPTION**  
496                   *Conformance for mlockall: PASS, NO\_OPTION*

497   **22**           **IF PCTS\_mlockall THEN**  
498                   **IF: PCTS\_RAP\_mlockall PCTS\_DETECT\_NO\_AP THEN:**  
499                   **TEST:**    A call to *mlockall()*, when the calling process does not have the appropriate  
500                                    privilege to perform the requested operation, returns a value of -1 and sets  
501                                    *errno* to [EPERM].  
502                   **ELSE NO\_TEST\_SUPPORT**  
503                   **ELSE NO\_OPTION**  
504                   *Conformance for mlockall: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

505   **12.1.2**       **Lock/Unlock a Range of Process Address Space**

506   Functions: *mlock()*, *munlock()*

507   **12.1.2.1**    **Synopsis**

508   **1**

509                   *M\_GA\_stdC\_proto\_decl(int; mlock; const void \*addr, size\_t len; sys/mman.h;;)*  
510                   **SEE:**    Assertion GA\_stdC\_proto\_decl in §2.7.3  
511                   *Conformance for mlock: PASS[1, 2], NO\_OPTION*

512   **2**

513                   *M\_GA\_commonC\_int\_result\_decl(mlock; sys/mman.h;;)*  
514                   **SEE:**    Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
515                   *Conformance for mlock: PASS[1, 2], NO\_OPTION*

- 516     **3**  
517     *M\_GA\_macro\_result\_decl(int; mlock; sys/mman.h;;)*  
518     **SEE:**   Assertion GA\_macro\_result\_decl in §1.3.4  
519     Conformance for *sem\_init*: *PASS, NO\_OPTION*
- 520     **4**  
521     *M\_GA\_macro\_args ( mlock; sys/mman.h;;)*  
522     **SEE:**   Assertion GA\_macro\_args in §2.7.3  
523     Conformance for *mlock*: *PASS, NO\_OPTION*
- 524     **5**  
525     *M\_GA\_stdC\_proto\_decl(int; munlock; const void \*addr, size\_tlen; sys/mman.h;;)*  
526     **SEE:**   Assertion GA\_stdC\_proto\_decl in §2.7.3  
527     Conformance for *munlock*: *PASS[5, 6], NO\_OPTION*
- 528     **6**  
529     *M\_GA\_commonC\_int\_result\_decl(munlock; const void \*addr, size\_tlen;*  
530     *sys/mman.h;; )*  
531     **SEE:**   Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
532     Conformance for *munlock*: *PASS[5, 6], NO\_OPTION*
- 533     **7**  
534     *M\_GA\_macro\_result\_decl(int; munlock; sys/mman.h;;)*  
535     **SEE:**   Assertion GA\_macro\_result\_decl in §1.3.4  
536     Conformance for *munlock*: *PASS, NO\_OPTION*
- 537     **8**  
538     *M\_GA\_macro\_args ( munlock; sys/mman.h;;)*  
539     **SEE:**   Assertion GA\_macro\_args in §2.7.3  
540     Conformance for *munlock*: *PASS, NO\_OPTION*

#### 541     **12.1.2.2 Description**

- 542     **mlock**   **FOR:**   *execl()*, *execv()*, *execle()*, *execve()*, *execlp()* and *execvp()*  
543     **IF** *PCTS\_MLOCK* **THEN**  
544         **IF** *PCTS\_GAP\_mlock* **THEN**  
545             **TEST:**   A successful call to the function *mlock(addr, len)* returns a value  
546             of zero and causes those whole pages containing any part of the  
547             address space of the process starting at address *addr* and  
548             continuing for *len* bytes to be memory resident until unlocked or  
549             until the process exits or executes a successful call to *function()*.  
550             **TR:**       Test for each of the three conditions.  
551             **NOTE:**   The assertion is tested once for each function specified in the FOR  
552             clause. The assertion is to be read by substituting *function()* with  
553             the current function specified in the FOR clause. The name of the  
554             function also is to be substituted for each occurrence in the  
555             construct *PCTS\_function*.  
556             **ELSE** *NO\_TEST\_SUPPORT*  
557             **ELSE** *NO\_OPTION*  
558             Conformance for *mlock*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

559 **D\_1** **IF** a *PCTS\_mlock* and a PCD.1b documents the following **THEN**  
560 **TEST:** A PCD.1b that documents the argument *addr.*,in a call to *mlock (addr,*  
561 *len)*, must be a multiple of the page size {PAGESIZE} does so in  
562 §12.1.2.2.  
563 **ELSE NO\_OPTION**  
564 *Conformance for mlock: PASS, NO\_OPTION*

565 **munlock**  
566 **IF** *PCTS\_munlock* **THEN**  
567 **IF** *PCTS\_mlock* and *PCTS\_GAP\_mlock* **THEN**  
568 **TEST:** The call *munlock (addr, len)* unlocks those whole pages containing any part of  
569 the address space of the process, with respect to the address space of the  
570 process, starting at address *addr* and continuing by *len* bytes, regardless of how  
571 many times *mlock()* has been called by the process for any of the pages in the  
572 specified ranges, and returns zero.  
573 **ELSE NO\_TEST\_SUPPORT**  
574 **ELSE NO\_OPTION**  
575 *Conformance for munlock: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

576 **D\_2** **IF** *PCTS\_munlock* documents the following **THEN**  
577 **TEST:** A PCD.1b that documents the argument *addr.*,in a call to *munlock*  
578 *(addr, len)*, must be a multiple of the page size {PAGESIZE} DOES  
579 so in §12.1.2.2.  
580 **ELSE NO\_OPTION**  
581 *Conformance for munlock: PASS, NO\_OPTION*

582 **9** **IF** *PCTS\_munlock* **THEN**  
583 **TEST:** When any of the pages in the range specified to a call to *munlock()* are  
584 also mapped into the address spaces of other processes, any locks  
585 established on those pages by another process are unaffected by the call  
586 of this process to *munlock()*.  
587 **ELSE NO\_OPTION**  
588 *Conformance for munlock: PASS, NO\_OPTION*

589 **10** **IF** *PCTS\_munlock* **THEN**  
590 **TEST:** When any of the pages in the range specified to a call to *munlock()* are  
591 also mapped into the address spaces of other processes, any locks  
592 established on those pages via the other mappings are unaffected by  
593 this call.  
594 **ELSE NO\_OPTION**  
595 *Conformance for munlock: PASS, NO\_OPTION*

596 **D\_3** **IF** *PCTS\_mlock* and PCD.1b documents the following **THEN**  
597 **TEST:** A PCD.1b that documents memory residency of unlocked pages does so  
598 in §12.1.2.2.  
599 **ELSE NO\_OPTION**  
600 *Conformance for mlock: PASS, NO\_OPTION*

601 **11** **IF** *PCTS\_mlock* **THEN**  
602 **IF** *PCTS\_RAP\_mlock* **THEN**  
603 **TEST:** Appropriate privilege is required to lock process memory with *mlock()*.  
604 **ELSE NO\_TEST\_SUPPORT**

605           **ELSE NO\_OPTION**  
606           *Conformance for mlock: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

607   **D\_4**   **IF** *PCTS\_mlock* and PCD.1b documents the following **THEN**  
608           **TEST:** A PCD.1b that documents whether or not it supports the *mlock()*  
609                   function does so in §12.1.2.2.  
610           **ELSE NO\_OPTION**  
611           *Conformance for mlock: PASS, NO\_OPTION*

612   **D\_5**   **IF** *PCTS\_munlock* and PCD.1b documents the following **THEN**  
613           **TEST:** A PCD.1b that documents whether or not it supports the *munlock()* does  
614                   so in §12.1.2.2.  
615           **ELSE NO\_OPTION**  
616           *Conformance for mlock: PASS, NO\_OPTION*

617   **12.1.2.3 Returns**

618   **R\_1**   **IF** *PCTS\_mlock* **THEN**  
619           **TEST:** When a call to *mlock()* completes successfully, the interface returns a  
620                   value of 0.  
621           **ELSE NO\_OPTION**  
622           **SEE:** Assertion *mlock* in §12.1.2.2

623   **R\_2**   **IF** *PCTS\_mlock* **THEN**  
624           **TEST:** When a call to *mlock()* completes successfully, the interface returns a  
625                   value of -1, sets *errno* to indicate the error, and no change is made to  
626                   any locks in the address space.  
627                   *PCTS\_mlock*  
628           **ELSE NO\_OPTION**  
629           **SEE:** All assertions in §12.1.2.4 controlled by *PCTS\_mlock*

630   **R\_3**   **IF** *PCTS\_munlock* **THEN**  
631           **TEST:** When a call to *munlock()* completes successfully, the interface returns  
632                   a value of 0.  
633           **ELSE NO\_OPTION**  
634           **SEE:** Assertion *munlock* in §12.1.2.2

635   **R\_4**   **IF** *PCTS\_munlock* **THEN**  
636           **TEST:** When a call to *munlock()* completes unsuccessfully, the interface  
637                   returns a value of -1, sets *errno* to indicate the error, and no change is  
638                   made to any locks in the address space.  
639                   *PCTS\_munlock*  
640           **ELSE NO\_OPTION**  
641           **SEE:** All assertions in §12.1.2.4 controlled by *PCTS\_munlock*

642   **12.1.2.4 Errors**

643   **12**   **IF** *PCTS\_mlock* **THEN**  
644           **IF:** *PCTS\_GAP\_mlock* **THEN**  
645           **TEST:** When a call to *mlock()*, when some or all of the address range specified by the  
646                   *addr* and *len* arguments does not correspond to valid mapped pages in the

647 address space of the process, returns a value of -1 and sets *errno* to  
648 [ENOMEN].  
649 **ELSE NO\_TEST\_SUPPORT**  
650 **ELSE NO\_OPTION**  
651 *Conformance for mlock: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

652 **13 IF PCTS\_munlock THEN**  
653 **TEST:** When a call to *munlock()*, when some or all of the address range specified by  
654 the *addr* and *len* arguments does not correspond to valid mapped pages in the  
655 address space of the process, returns a value of -1 and sets *errno* to  
656 [ENOMEN].  
657 **ELSE NO\_OPTION**  
658 *Conformance for munlock: PASS, NO\_OPTION*

659 **14 IF not PCTS\_mlock THEN**  
660 **TEST:** A call to *mlock()*, returns a value of -1 and sets *errno* to [ENOSYS].  
661 **ELSE NO\_OPTION**  
662 *Conformance for mlock: PASS, NO\_OPTION*

663 **15 IF not PCTS\_munlock THEN**  
664 **TEST:** A call to *munlock()*, returns a value of -1 and sets *errno* to [ENOSYS].  
665 **ELSE NO\_OPTION**  
666 *Conformance for mlock: PASS, NO\_OPTION*

667 **16 IF PCTS\_mlock THEN**  
668 **IF: PCTS\_GAP\_mlock THEN**  
669 **TEST:** A call to *mlock()*, when some or all of the memory identified by the operation  
670 could not be locked, returns a value of -1 and sets *errno* to [EAGAIN].  
671 **ELSE NO\_TEST\_SUPPORT**  
672 **ELSE NO\_OPTION**  
673 *Conformance for mlock: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

674 **D\_6 IF PCTS\_mlock or PCTS\_munlock and a PCD.1b documents the following THEN**  
675 **TEST:** A PCD.1b that documents the implementation requires memory locking  
676 only in multiples of {PAGESIZE} does so in §12.1.2.4.  
677 **ELSE NO\_OPTION**  
678 *Conformance for mlock: PASS, NO\_OPTION*

679 **17 IF PCTS\_mlock and PCTS\_MULTIPLE\_OF\_PAGESIZE and**  
680 **PCTS\_DETECT\_NOT\_MULTIPLE\_OF\_PAGESIZE THEN**  
681 **IF PCTS\_GAP\_mlock THEN**  
682 **TEST:** A call to *mlock()*, when the *addr* argument is not a multiple of the  
683 page size {PAGESIZE}, returns a value of -1 and sets *errno* to  
684 [EINVAL].  
685 **ELSE NO\_TEST\_SUPPORT**  
686 **ELSE NO\_OPTION**  
687 *Conformance for mlock: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

688 **18 IF PCTS\_MULTIPLE\_OF\_PAGESIZE and PCTS\_DETECT\_NOT\_MULTIPLE\_OF\_PAGESIZE and**  
689 **PCTS\_munlock THEN**  
690 **TEST:** A call to *munlock()*, when the *addr* argument is not a multiple of the  
691 page size {PAGESIZE}, returns a value of -1 and sets *errno* to  
692 [EINVAL].  
693 **ELSE NO\_OPTION**  
694 *Conformance for mlock: PASS,, NO\_OPTION*

695 **D\_7** **IF** *PCTS\_mlock* and a PCD.1b documents the following **THEN**  
696 **TEST:** A PCD.1b that documents the implementation defined limit on an  
697 amount of memory that a process may lock does so in §12.1.2.4.  
698 **ELSE NO\_OPTION**  
699 *Conformance for mlock: PASS, NO\_OPTION*

700 **19** **IF** *PCTS\_mlock* and *PCTS\_DETECT\_LOCKABLE\_MEMORY\_LIMIT\_mlock* **THEN**  
701 **IF** *PCTS\_GAP\_mlock* **THEN**  
702 **TEST:** A call to *mlock()*, when locking the pages mapped by the specified  
703 range would exceed an implementation-defined limit on the  
704 amount of memory that the process may lock, returns a value of  
705 -1 and sets *errno* to [ENOMEM].  
706 **ELSE NO\_TEST\_SUPPORT**  
707 **ELSE NO\_OPTION**  
708 *Conformance for mlock: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

709 **D\_8** **IF** *PCTS\_mlock* and a PCD.1b documents the following **THEN**  
710 **TEST:** A PCD.1b that documents the implementation detects whether or not a  
711 process has appropriate privileges to lock pages does so in §12.1.2.4.  
712 **ELSE NO\_OPTION**  
713 *Conformance for mlock: PASS, NO\_OPTION*

714 **20** **IF** *PCTS\_mlock* and *PCTS\_DETECT\_NO\_AP* **THEN**  
715 **IF** *PCTS\_RAP\_mlock* **THEN**  
716 **TEST:** A call to *mlock()*, when the calling process does not have the  
717 appropriate privilege to perform the requested operation , returns  
718 a value of -1 and sets *errno* to [EPERM].  
719 **ELSE NO\_TEST\_SUPPORT**  
720 **ELSE NO\_OPTION**  
721 *Conformance for mlock: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

## 722 12.2 Memory Mapping Functions

### 723 12.2.1 Map Process Address to a Memory Object

724 Function: *mmap()*

725 **1**  
726 *M\_GA\_stdC\_proto\_decl(void \*; mmap; void\*addr, size\_tlen; int prot, int flags, int*  
727 *fildes, off\_t off; sys/mman.h;;;)*  
728 **SEE:** Assertion *GA\_stdC\_proto\_decl* in §2.7.3  
729 *Conformance for mlock: PASS[1, 2], NO\_OPTION*

730 **2**  
731 *M\_GA\_commonC\_result\_decl ( void\*; mmap; sys/mman.h;;;)*  
732 **SEE:** Assertion *GA\_commonC\_int\_result\_decl* in §2.7.3  
733 *Conformance for mlock: PASS[1, 2], NO\_OPTION*

734 **3**



735 *M\_GA\_macro\_result\_decl(int; mlock; sys/mman.h;;;)*  
 736 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 737 *Conformance for sem\_init: PASS, NO\_OPTION*

738 **4**  
 739 *M\_GA\_macro\_args ( mmap; sys/mman.h;;;)*  
 740 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 741 *Conformance for mlock: PASS, NO\_OPTION*

742 **mmap** **IF** *PCTS\_mmap* **THEN**  
 743 **TEST:** A successful call to the function *mmap()* establishes a mapping  
 744 between the address space of the process for *len* bytes to the memory  
 745 object represented by the file descriptor *fildes* at offset *off* for *len* bytes,  
 746 and returns the address at which the mapping was placed.  
 747 **ELSE** *NO\_OPTION*  
 748 *Conformance for mmap: PASS, NO\_OPTION*

749 **D\_1** **IF** *PCTS\_mmap* and a PCD.1b documents the following **THEN**  
 750 **TEST:** A PCD.1b that documents the implementation-dependent function of the  
 751 parameter *addr* and the values of *flags*, which determines the address  
 752 at which the mapping is placed, does so in §12.1.2.2.  
 753 **ELSE** *NO\_OPTION*  
 754 *Conformance for mmap: PASS, NO\_OPTION*

755 **5** **IF** *PCTS\_mmap* **THEN**  
 756 **TEST:** The address range starting at the address returned by *mmap(addr, len,*  
 757 *prot, flags, fildes, off)*, and continuing for *len* bytes, is legitimate for  
 758 the possible (not necessarily current) address space of the process.  
 759 **ELSE** *NO\_OPTION*  
 760 *Conformance for mmap: PASS, NO\_OPTION*

761 **6** **IF** *PCTS\_mmap* **THEN**  
 762 **TEST:** In a call to *mmap(addr, len, prot, flags, fildes, off)* the ranges of bytes  
 763 starting at *off* and continuing for *len* bytes is legitimate for the possible  
 764 (not necessarily current) offsets in the file or shared memory object  
 765 represented by *fildes*..  
 766 **ELSE** *NO\_OPTION*  
 767 *Conformance for mmap: PASS, NO\_OPTION*

768 **7** **IF** *PCTS\_mmap* **THEN**  
 769 **TEST:** The mapping established by *mmap(addr, len, prot, flags, fildes,*  
 770 *off)*, replaces any previous mappings for those whole pages containing  
 771 any part of the address space of the process, starting at the address  
 772 returned by *mmap()* and continuing for *len* bytes.  
 773 **ELSE** *NO\_OPTION*  
 774 *Conformance for mmap: PASS, NO\_OPTION*

775 **8** **IF** *PCTS\_mmap* **THEN**

776                   **TEST:** In a call to *mmap(addr, len, prot, flags, fildes, off)*, the parameter *prot*  
 777                   determines whether read, write, execute, or some combination of  
 778                   accesses are permitted to the data being mapped.

779                   **ELSE NO\_OPTION**  
 780                   Conformance for *mmap*: *PASS, NO\_OPTION*

## 781 **prot\_values**

782                   **IF PCTS\_mmap THEN**  
 783                   **SETUP:** Include the header `<sys/mman.h>`  
 784                   **TEST:** The constants `PROT_NONE`, `PROT_READ`, `PROT_WRITE`, and `PROT_EXEC`. are defined and  
 785                   are bitwise distinct.  
 786                   **ELSE NO\_OPTION**  
 787                   Conformance for *mmap*: *PASS, NO\_OPTION*

788                   **9 IF PCTS\_mmap THEN**  
 789                   **TEST:** In the call *mmap(addr, len, prot, flags, fildes, off)*, the argument *prot* may be either  
 790                   `PROT_NONE`, or the bitwise inclusive **OR** of one or more of the flags `PROT_READ`,  
 791                   `PROT_WRITE` `PROT_EXEC`. and `PROT_NONE`.  
 792                   **TR:** Try `PROT_NONE` and all eight bitwise combinations of the other three values.  
 793                   **ELSE NO\_OPTION**  
 794                   Conformance for *mmap*: *PASS, NO\_OPTION*

795                   **R\_1 IF PCTS\_mmap THEN**  
 796                   **TEST:** When the combination of access types specified by *prot* is not supported,  
 797                   the call *mmap(addr, len, prot, flags, fildes, off)* fails.  
 798                   **ELSE NO\_OPTION**  
 799                   **SEE:** Assertion `mmap_ENOTSUP` in §12.1.2.2

800                   **10 IF PCTS\_mmap THEN**  
 801                   **IF {POSIX\_MEMORY\_PROTECTION} THEN**  
 802                   **TEST:** When `PROT_WRITE` is unset, writes to the region mapped by *mmap()*  
 803                   fail.  
 804                   **ELSE NO\_TEST\_SUPPORT**  
 805                   **ELSE NO\_OPTION**  
 806                   Conformance for *mmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

## 807 **mem\_protect\_flags**

808                   **11 IF PCTS\_mmap THEN**  
 809                   **IF {POSIX\_MEMORY\_PROTECTION} THEN**  
 810                   **TEST:** When `PROT_NONE` alone has been set, writes to the region mapped by  
 811                   *mmap(addr, len, prot, flags, fildes, off)* fail.  
 812                   **ELSE NO\_TEST\_SUPPORT**  
 813                   **ELSE NO\_OPTION**  
 814                   Conformance for *mmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

## 815 **mem\_protect\_flags**

816                   **IF PCTS\_mmap THEN**  
 817                   **IF {POSIX\_MEMORY\_PROTECTION} THEN**

818                   **TEST:** The implementation supports the following values of *prot*:  
819                   PROT\_NONE, PROT\_READ, PROT\_WRITE, and the inclusive OR of PROT\_READ  
820                   and PROT\_WRITE  
821                   **ELSE NO\_TEST\_SUPPORT**  
822                   **ELSE NO\_OPTION**  
823                   Conformance for mmap: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

824   **12**       **IF PCTS\_mmap THEN**  
825               **TEST:** after a call to *mmap(addr, len, prot, flags, fildes, off)* with the flag  
826               MAP\_SHARED set, write references change the underlying object.  
827               **ELSE NO\_OPTION**  
828               Conformance for mmap: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

829   **13**       **IF PCTS\_mmap THEN**  
830               **IF PCTS\_MAP\_PRIVATE THEN**  
831               **TEST:** After a call to *mmap( addr, len, prot, flags, fildes, off)*, with the  
832               flag MAP\_PRIVATE set, modifications to the mapped data by the  
833               calling process are visible only to the calling process and do not  
834               change the underlying object.  
835               **ELSE NO\_TEST\_SUPPORT**  
836               **ELSE NO\_OPTION**  
837               Conformance for mmap: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

838   **D\_2**       **IF PCTS\_mmap** and a PCD.1b documents the following **THEN**  
839               **TEST:** A PCD.1b that documents whether modifications to the underlying  
840               object, done after the MAP\_PRIVATE mapping is established, are visible  
841               through the MAP\_PRIVATE mapping, does so in §12.1.2.2.  
842               **ELSE NO\_OPTION**  
843               Conformance for mmap: PASS, NO\_OPTION

844   **14**       **IF PCTS\_mmap THEN**  
845               **IF PCTS\_MAP\_PRIVATE THEN**  
846               **TEST:** MAP\_SHARED must be specified in a call to the *mmap()* function.  
847               **ELSE NO\_TEST\_SUPPORT**  
848               **ELSE NO\_OPTION**  
849               Conformance for mmap: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

850   **15**       **IF PCTS\_mmap THEN**  
851               **TEST:** the mapping type is retained across *fork()*.  
852               **ELSE NO\_OPTION**  
853               Conformance for mmap: PASS, NO\_OPTION

854   **16**       **IF PCTS\_mmap THEN**  
855               **IF PCTS\_MAP\_FIXED THEN**  
856               **TEST:** When MAP\_FIXED is set, the address returned by *mmap( addr, len,*  
857               *prot, flags, fildes, off)* is *addr* exactly..  
858               **ELSE NO\_TEST\_SUPPORT**  
859               **ELSE NO\_OPTION**

860 *Conformance for mmap: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

861 **D\_3** **IF** *PCTS\_mmap* and a PCD.1b documents the following **THEN**  
862 **TEST:** A PCD.1b that documents whether `MAP_FIXED` is supported in §12.1.2.2.  
863 **ELSE** *NO\_OPTION*  
864 *Conformance for mmap: PASS, NO\_OPTION*

865 **D\_4** **IF** *PCTS\_mmap* **THEN**  
866 **TEST:** A PCD.1b documents how the system uses *addr* to arrive at *pa*, when  
867 `MAP_FIXED` is not set in §12.1.2.2.  
868 **ELSE** *NO\_OPTION*  
869 *Conformance for mmap: PASS, NO\_OPTION*

870 **17** **IF** *PCTS\_mmap* **THEN**  
871 **TEST:** When `MAP_FIXED` is not set, a call to *mmap()* never places a mapping at  
872 address zero.  
873 **ELSE** *NO\_OPTION*  
874 *Conformance for mmap: PASS,, NO\_OPTION*

875 **18** **IF** *PCTS\_mmap* **THEN**  
876 **TEST:** When `MAP_FIXED` is not set, a call to *mmap()* never replaces an extant  
877 mapping.  
878 **ELSE** *NO\_OPTION*  
879 *Conformance for mmap: PASS,, NO\_OPTION*

880 **19** **IF** *PCTS\_mmap* **THEN**  
881 **IF** *PCTS\_MAP\_FIXED* **THEN**  
882 **TEST:** When `MAP_FIXED` is specified and *addr* is nonzero, the address  
883 returned by *mmap(addr, len, prot, flags, fildes, off)* has the same  
884 remainder as the *off* parameter, modulo the page size  
885 {PAGESIZE}.  
886 **ELSE** *NO\_TEST\_SUPPORT*  
887 **ELSE** *NO\_OPTION*  
888 *Conformance for mmap: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

889 **20** **IF** *PCTS\_mmap* **THEN**  
890 **IF** *PCTS\_MULTIPLE\_OF\_PAGESIZE* **THEN**  
891 **TEST:** The argument *off* must be a multiple of the page size..  
892 **ELSE** *NO\_TEST\_SUPPORT*  
893 **ELSE** *NO\_OPTION*  
894 *Conformance for mmap: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

895 **D\_5** **IF** *PCTS\_mmap* and a PCD.1b documents the following **THEN**  
896 **TEST:** A PCD.1b that documents that the argument *off* , in a call to  
897 *mmap(addr, len, prot, flags, fildes, off)* must be a multiple of the page  
898 size, does so in §12.1.2.2.  
899 **ELSE** *NO\_OPTION*  
900 *Conformance for mmap: PASS, NO\_OPTION*

901     **21**     **IF** *PCTS\_mmap* **THEN**  
902             **IF** *PCTS\_MULTIPLE\_OF\_PAGESIZE* and *PCTS\_MAP\_FIXED* **THEN**  
903                 **TEST:** When *MAP\_FIXED* is specified, the argument *addr* must be a  
904                     multiple of the page size.  
905                 **ELSE** *NO\_TEST\_SUPPORT*  
906             **ELSE** *NO\_OPTION*  
907             Conformance for *mmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

908     **D\_6**     **IF** *PCTS\_mmap* and a PCD.1b documents the following **THEN**  
909             **TEST:** A PCD.1b that documents that the argument *addr* , in a call to  
910                     *mmap(addr, len, prot, flags, fildes, off)* with *MAP\_FIXED* specified,  
911                     must be a multiple of the page size, does so in §12.1.2.2.  
912             **ELSE** *NO\_OPTION*  
913             Conformance for *mmap*: *PASS, NO\_OPTION*

914     **22**     **IF** *PCTS\_mmap* **THEN**  
915             **IF** *PCTS\_MAP\_FIXED* **THEN**  
916                 **TEST:** When *MAP\_FIXED* is specified, the parameter *len* need not meet a  
917                     size or alignment constraint.  
918                 **ELSE** *NO\_TEST\_SUPPORT*  
919             **ELSE** *NO\_OPTION*  
920             Conformance for *mmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

921     **23**     **IF** *PCTS\_mmap* **THEN**  
922             **IF** *PCTS\_MAP\_FIXED* **THEN**  
923                 **TEST:** When *MAP\_FIXED* is specified, after a call to *mmap( addr, len,*  
924                     *prot, flags, fildes, off)* any partial page specified by the address  
925                     range starting at the returned address and continuing for *len* bytes  
926                     is included in the object.  
927                 **ELSE** *NO\_TEST\_SUPPORT*  
928             **ELSE** *NO\_OPTION*  
929             Conformance for *mmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

930     **24**     **IF** *PCTS\_mmap* **THEN**  
931             **TEST:** Any partial page at the end of an object is zero-filled.  
932             **ELSE** *NO\_OPTION*  
933             Conformance for *mmap*: *PASS, NO\_OPTION*

934     **25**     **IF** *PCTS\_mmap* **THEN**  
935             **TEST:** Modified portions of the last page of an object that are beyond its end  
936                     are not written out.  
937             **ELSE** *NO\_OPTION*  
938             Conformance for *mmap*: *PASS, NO\_OPTION*

939     **mmap\_SIGBUS**  
940             **IF** *PCTS\_mmap* **THEN**  
941                 **IF** {*POSIX\_MEMORY\_PROTECTION*} **THEN**  
942                     **TEST:** Following a call to *mmap( addr, len, prot, flags, fildes, off),*  
943                     references within the address range starting at the returned address

944 and continuing for *len* bytes to whole pages following the end of an  
 945 object result in delivery of a SIGBUS signal.

946 **ELSE NO\_TEST\_SUPPORT**

947 **ELSE NO\_OPTION**

948 *Conformance for mmap: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

949 **D\_7 IF PCTS\_mmap** and a PCD.1b documents the following **THEN**

950 **TEST:** A PCD.1b that documents the result of references within the address  
 951 range starting at *pa* and continuing for *len* bytes, to whole pages  
 952 following the end of an object, when the  
 953 {\_POSIX\_MEMORY\_PROTECTION} option is not supported, does so in  
 954 §12.1.2.2.

955 **ELSE NO\_OPTION**

956 *Conformance for mmap: PASS, NO\_OPTION*

957 **D\_8 IF PCTS\_mmap** and a PCD.1b documents the following **THEN**

958 **TEST:** A PCD.1b that documents whether or not it supports the *mmap()*  
 959 function, does so in §12.1.2.2.

960 **ELSE NO\_OPTION**

961 *Conformance for mmap: PASS, NO\_OPTION*

### 962 **12.2.1.3 Returns**

963 **R\_2 IF PCTS\_mmap THEN**

964 **TEST:** When a call to *mmap()* completes successfully, the interface returns the  
 965 address at which the mapping was placed.

966 **ELSE NO\_OPTION**

967 **SEE:** Assertion *mmap* in §12.2.1.2

968 **R\_3 IF PCTS\_mmap THEN**

969 **TEST:** When a call to *mmap()* completes unsuccessfully, the interface returns  
 970 a value of *MAP\_FAILED*, and sets *errno* to indicate the error

971 **ELSE NO\_OPTION**

972 **SEE:** Assertion *mmap* in §12.2.1.4

973 **26 IF PCTS\_mmap THEN**

974 **SETUP:** Include the header `<sys/mman.h>`

975 **TEST:** The symbol *MAP\_FAILED* is defined.

976 **ELSE NO\_OPTION**

977 *Conformance for mmap: PASS, NO\_OPTION*

978 **R\_4 IF PCTS\_mmap THEN**

979 **TEST:** No successful return from *mmap()* returns the value of *MAP\_FAILED*.

980 **ELSE NO\_OPTION**

981 **SEE:** Assertion *mmap* in §12.2.1.4

### 982 **12.2.1.4 Errors**

983 **27 IF PCTS\_mmap THEN**

- 984                   **TEST:** A call to *mmap( addr, len, prot, flags, fildes, off)*, when the file  
 985                   descriptor *fildes* is not open for read, returns a value of `MAP_FAILED`  
 986                   and sets *errno* to `[EACCESS]`.  
 987                   **TR:** Specify each possible protection  
 988                   **ELSE NO\_OPTION**  
 989                   Conformance for *mmap*: *PASS, NO\_OPTION*
- 990   **28**           **IF PCTS\_mmap THEN**  
 991                   **TEST:** A call to *mmap( addr, len, prot, flags, fildes, off)*, when the file  
 992                   descriptor *fildes* is not open for write and `PROT_WRITE` is specified for  
 993                   a `MAP_SHARED` type mapping, returns a value of `MAP_FAILED` and sets  
 994                   *errno* to `[EACCESS]`.  
 995                   **ELSE NO\_OPTION**  
 996                   Conformance for *mmap*: *PASS, NO\_OPTION*
- 997   **29**           **IF PCTS\_mmap THEN**  
 998                   **TEST:** A call to *mmap()*, when the mapping could not be locked in memory, if  
 999                   required by *mlockall()*, due to a lack of resources, returns a value of  
 1000                   `MAP_FAILED` and sets *errno* to `[EAGAIN]`.  
 1001                   **ELSE NO\_OPTION**  
 1002                   Conformance for *mmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 1003   **30**           **IF PCTS\_mmap THEN**  
 1004                   **TEST:** A call to *mmap( addr, len, prot, flags, fildes, off)*, when the *fildes*  
 1005                   argument is not a valid open file descriptor, returns a value of  
 1006                   `MAP_FAILED` and sets *errno* to `EBADF`.  
 1007                   **ELSE NO\_OPTION**  
 1008                   Conformance for *mmap*: *PASS, NO\_OPTION*
- 1009   **31**           **IF PCTS\_mmap THEN**  
 1010                    **IF PCTS\_DETECT\_INVALID\_FLAGS\_mmap THEN**  
 1011                      **TEST:** Following a call to *mmap( addr, len, prot, flags, fildes, off)*, when  
 1012                      the value in *flags* is invalid (e.g., neither `MAP_PRIVATE` or  
 1013                      `MAP_SHARED` is set) returns a value of `MAP_FAILED` and sets *errno*  
 1014                      to `[EINVAL]`.  
 1015                    **ELSE NO\_TEST\_SUPPORT**  
 1016                   **ELSE NO\_OPTION**  
 1017                   Conformance for *mmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 1018   **32**           **IF PCTS\_mmap THEN**  
 1019                   **TEST:** A call to *mmap( addr, len, prot, flags, fildes, off)*, when the *fildes*  
 1020                   argument refers to an object for which *mmap()* is meaningless, such as  
 1021                   a terminal, returns a value of `MAP_FAILED` and sets *errno* to  
 1022                   `[ENODEV]`.  
 1023                   **ELSE NO\_OPTION**  
 1024                   Conformance for *mmap*: *PASS, NO\_OPTION*
- 1025   **33**           **IF PCTS\_mmap THEN**  
 1026                    **IF PCTS\_MAP\_FIXED THEN**

1027                   **TEST:** A call to *mmap( addr, len, prot, flags, fildes, off)*, when  
1028                   MAP\_FIXED is specified, and the address range starting at *addr* and  
1029                   continuing for *len* bytes exceeds that allowed for the address space  
1030                   of a process; or MAP\_FIXED is not specified, and there is  
1031                   insufficient room in the address space to effect the mapping; returns  
1032                   a value of MAP\_FAILED and sets *errno* to [ENOMEM].  
1033                   There is no known reliable test method for this assertion.  
1034                   **ELSE NO\_TEST\_SUPPORT**  
1035                   **ELSE NO\_OPTION**  
1036                   Conformance for *mmap*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1037   **34**   **IF PCTS\_mmap THEN**  
1038            **IF PCTS\_mlockall PCTS\_GAP\_mlockall THEN**  
1039            **TEST:** A call to *mmap()*, when the mapping could not be locked in  
1040            memory, if required by *mlockall()*, because it would require more  
1041            space than the system is able to supply, returns a value of  
1042            MAP\_FAILED and sets *errno* to [ENOMEM].  
1043            **ELSE NO\_TEST\_SUPPORT**  
1044            **ELSE NO\_OPTION**  
1045            Conformance for *mmap*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1046   **35**   **IF not PCTS\_mmap THEN**  
1047            **TEST:** A call to *mmap()*, returns a value of MAP\_FAILED and sets *errno* to  
1048            [ENOSYS].  
1049            **ELSE NO\_OPTION**  
1050            Conformance for *mmap*: PASS,, NO\_OPTION

1051   **36**   **IF PCTS\_mmap THEN**  
1052            **IF not PCTS\_MAP\_FIXED THEN**  
1053            **TEST:** A call to *mmap( addr, len, prot, flags, fildes, off)*, when  
1054            MAP\_FIXED is specified in the *flags* argument, returns a value of  
1055            MAP\_FAILED and sets *errno* to [ENOTSUP].  
1056            **ELSE NO\_TEST\_SUPPORT**  
1057            **ELSE NO\_OPTION**  
1058            Conformance for *mmap*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1059   **37**   **IF PCTS\_mmap THEN**  
1060            **IF not PCTS\_MAP\_PRIVATE THEN**  
1061            **TEST:** A call to *mmap( addr, len, prot, flags, fildes, off)*, when  
1062            MAP\_PRIVATE is specified in the *flags* argument, a value of  
1063            MAP\_FAILED and sets *errno* to [ENOTSUP].  
1064            **ELSE NO\_TEST\_SUPPORT**  
1065            **ELSE NO\_OPTION**  
1066            Conformance for *mmap*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1067   **mmap\_ENOTSUP**  
1068            **IF PCTS\_mmap THEN**



1069                   **TEST:** A call to *mmap()*, when the implementation does not support the  
 1070   combination of accesses requested in the *prot* argument, returns a value  
 1071   of `MAP_FAILED` and sets *errno* to `[ENOTSUP]`.

1072                   **TR:** Test with `MAP_SHARED` and `MAP_PRIVATE` both in the *prot* argument.

1073                   **ELSE NO\_TEST\_SUPPORT**

1074                   **ELSE NO\_OPTION**

1075                   Conformance for *mmap*: *PASS,, NO\_OPTION*

1076   **38**           **IF PCTS\_mmap THEN**

1077                   **TEST:** A call to *mmap(addr, len, prot, flags, fildes, off)*, when the addresses in  
 1078   the range starting at *off* and continuing for *len* bytes are invalid for the  
 1079   object specified by *fildes*, returns a value of `MAP_FAILED` and sets *errno*  
 1080   to `[ENXIO]`.

1081                   **ELSE NO\_TEST\_SUPPORT**

1082                   **ELSE NO\_OPTION**

1083                   Conformance for *mmap*: *PASS, NO\_OPTION*

1084   **39**           **IF PCTS\_mmap THEN**

1085                   **IF PCTS\_MAP\_FIXED THEN**

1086                   **TEST:** A call to *mmap(addr, len, prot, flags, fildes, off)*, when `MAP_FIXED`  
 1087   is specified in *flags* and the combination of *addr*, *len*, and *off* is  
 1088   invalid for the object specified by *fildes*, returns a value of  
 1089   `MAP_FAILED` and sets *errno* to `[ENXIO]`.

1090                   **ELSE NO\_TEST\_SUPPORT**

1091                   **ELSE NO\_OPTION**

1092                   Conformance for *mmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1093   **40**           **IF PCTS\_mmap THEN**

1094                   **IF PCTS\_MULTIPLE\_OF\_PAGESIZE THEN**

1095                   **TEST:** A call to *mmap(addr, len, prot, flags, fildes, off)*, when the  
 1096   argument *off* is not a multiple of the page size `{PAGESIZE}`,  
 1097   returns a value of `MAP_FAILED` and sets *errno* to `[EINVAL]`.

1098                   **ELSE NO\_TEST\_SUPPORT**

1099                   **ELSE NO\_OPTION**

1100                   Conformance for *mmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1101   **41**           **IF PCTS\_mmap THEN**

1102                   **IF PCTS\_MULTIPLE\_OF\_PAGESIZE and PCTS\_MAP\_FIXED THEN**

1103                   **TEST:** A call to *mmap(addr, len, prot, flags, fildes, off)*, when `MAP_FIXED`  
 1104   is specified and the argument *addr* is not a multiple of the page size  
 1105   `{PAGESIZE}`, returns a value of `MAP_FAILED` and sets *errno* to  
 1106   `[EINVAL]`.

1107                   **ELSE NO\_TEST\_SUPPORT**

1108                   **ELSE NO\_OPTION**

1109                   Conformance for *mmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

### 1110   **12.2.2.2 Unmap Previously Mapped Addresses**

1111   function: *munmap()*

1112 **12.2.2.1 Synopsis**1113 **1**1114 *M\_GA\_stdC\_proto\_decl( int;; munmap; void\*addr, size\_t len; sys/mman.h;;)*1115 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.31116 *Conformance for munmap: PASS[1, 2], NO\_OPTION*1117 **2**1118 *M\_GA\_commonC\_result\_decl ( munmap; sys/mman.h;;)*1119 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.31120 *Conformance for munmap: PASS[1, 2], NO\_OPTION*1121 **3**1122 *M\_GA\_macro\_result\_decl(int; munmap; sys/mman.h;;)*1123 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.41124 *Conformance for munmap: PASS, NO\_OPTION*1125 **4**1126 *M\_GA\_macro\_args ( munmap; sys/mman.h;;)*1127 **SEE:** Assertion GA\_macro\_args in §2.7.31128 *Conformance for munmap: PASS, NO\_OPTION*1129 **12.2.2.2 Description**1130 **munmap**1131 **IF PCTS\_munmap THEN**1132 **TEST:** A successful call to the function *munmap(addr, len)*, removes any  
1133 mappings for those entire pages containing any part of the address space  
1134 of the process starting at *addr* and continuing for *len* bytes, and returns  
1135 the value zero.1136 **ELSE NO\_OPTION**1137 *Conformance for munmap: PASS, NO\_OPTION*1138 **munmap\_SIGSEV**1139 **IF PCTS\_munmap THEN**1140 **TEST:** Following a successful call to *munmap()*, references to the unmapped  
1141 pages results in the delivery of a SIGSEV signal to the process.1142 **ELSE NO\_OPTION**1143 *Conformance for munmap: PASS, NO\_OPTION*1144 **5**1144 **IF PCTS\_munmap THEN**1145 **TEST:** When there are no mappings in the specified address range, then  
1146 *munmap()* has no effect.1147 **ELSE NO\_OPTION**1148 *Conformance for munmap: PASS, NO\_OPTION*1149 **6**1149 **IF PCTS\_munmap THEN**1150 **IF PCTS\_MULTIPLE\_OF\_PAGESIZE THEN**

1151                   **TEST:** In a call to *munmap(addr, len)*, the argument *addr* must be a  
 1152                   multiple of the page size, {PAGESIZE}.  
 1153                   **ELSE NO\_TEST\_SUPPORT**  
 1154                   **ELSE NO\_OPTION**  
 1155                   Conformance for *munmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1156 **D\_1**   **IF** *PCTS\_munmap* and a PCD.1b documents the following **THEN**  
 1157                   **TEST:** A PCD.1b that documents that the argument *addr*, in a call to  
 1158                   *munmap(addr, len)*, must be a multiple of the page size, {PAGESIZE},  
 1159                   does so in §12.2.2.2.  
 1160                   **ELSE NO\_OPTION**  
 1161                   Conformance for *munmap*: *PASS, NO\_OPTION*

1162 **7**       **IF** *PCTS\_munmap* **THEN**  
 1163                   **IF** *PCTS\_munmap* and *PCTS\_MAP\_PRIVATE* **THEN**  
 1164                   **TEST:** When a mapping to be removed is private, any modifications made  
 1165                   in this address range are discarded.  
 1166                   **ELSE NO\_TEST\_SUPPORT**  
 1167                   **ELSE NO\_OPTION**  
 1168                   Conformance for *munmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1169 **munlock\_remove\_maps**  
 1170                   **IF** *PCTS\_munmap* **THEN**  
 1171                   **IF** *PCTS\_mlock* and *PCTS\_GAP\_MLOCK* and *PCTS\_mlockall* and  
 1172                   *PCTS\_GAP\_MLOCKALL* and *PCTS\_munlock* **THEN**  
 1173                   **TEST:** Following a call to *munmap(addr, len)*, any memory locks (see  
 1174                   POSIX.1b {3} §12.1 2 and POSIX.1b {3} §12.1 1 is associated with  
 1175                   this address range are removed, as if by an appropriate call to  
 1176                   *munlock()*.  
 1177                   **ELSE NO\_TEST\_SUPPORT**  
 1178                   **ELSE NO\_OPTION**  
 1179                   Conformance for *munmap*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1180 **D\_2**   **IF** *PCTS\_munmap* and a PCD.1b documents the following **THEN**  
 1181                   **TEST:** A PCD.1b that documents the behavior of this function if the mapping  
 1182                   was not established by a call to *munmap()*, does so in §12.2.2.2.  
 1183                   **ELSE NO\_OPTION**  
 1184                   Conformance for *munmap*: *PASS, NO\_OPTION*

1185 **D\_3**   **IF** *PCTS\_munmap* and a PCD.1b documents the following **THEN**  
 1186                   **TEST:** A PCD.1b that documents whether or not it supports the *munmap()*  
 1187                   function does so in §12.2.2.2.  
 1188                   **ELSE NO\_OPTION**  
 1189                   Conformance for *munmap*: *PASS, NO\_OPTION*

### 1190 **12.2.2.3 Returns**

1191 **R\_1**   **IF** *PCTS\_munmap* **THEN**

1192                   **TEST:** When a call to *munmap()* completes successfully, the interface returns  
 1193                   to a value of 0.  
 1194                   **ELSE NO\_OPTION**  
 1195                   **SEE:** Assertion *munmap* in §12.2.2.2

1196 **R\_2**   **IF PCTS\_munmap THEN**  
 1197                   **TEST:** When a call to *munmap()* completes unsuccessfully, the interface returns  
 1198                   a value of -1 and sets *errno* to indicate the error.  
 1199                   **ELSE NO\_OPTION**  
 1200                   **SEE:** All assertions in §12.2.2.4

#### 1201 **12.2.2.4 Errors**

1202 **8**       **IF PCTS\_munmap THEN**  
 1203                   **TEST:** A call to *munmap()*, when some of the addresses in the range starting at  
 1204                   *addr* and continuing for *len* bytes are outside the range allowed for the  
 1205                   address space of a process, returns a value of -1 and sets *errno* to  
 1206                   [EINVAL].  
 1207                   **NOTE:** There is no known portable test method for this assertion.  
 1208                   **ELSE NO\_OPTION**  
 1209                   *Conformance for munmap: PASS, NO\_TEST, NO\_OPTION*

1210 **9**       **IF not PCTS\_munmap THEN**  
 1211                   **TEST:** A call to *munmap()*, returns a value of -1 and sets *errno* to [ENOSYS].  
 1212                   **ELSE NO\_OPTION**  
 1213                   *Conformance for munmap: PASS, NO\_OPTION*

1214 **10**      **IF PCTS\_munmap THEN**  
 1215                   **IF PCTS\_MULTIPLE\_OF\_PAGESIZE**  
 1216                   and **PCTS\_DETECT\_NOT\_MULTIPLE\_OF\_PAGESIZE THEN**  
 1217                   **TEST:** A call to *munmap(addr, len,)*, when the value of *addr* is not a  
 1218                   multiple of the page size {PAGESIZE}, returns a value of -1 and  
 1219                   sets *errno* to [EINVAL].  
 1220                   **ELSE NO\_TEST\_SUPPORT**  
 1221                   **ELSE NO\_OPTION**  
 1222                   *Conformance for munmap: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

#### 1223 **12.2.3 Change Memory Protection**

1224 Function: *mprotect()*

##### 1225 **12.2.3.1 Synopsis**

1226 **1**  
 1227 *M\_GA\_stdC\_proto\_decl( int; mprotect; const void\*addr, size\_t len; int\_prot;*  
 1228 *sys/mman.h;;)*  
 1229 **SEE:** Assertion *GA\_stdC\_proto\_decl* in §2.7.3  
 1230 *Conformance for mprotect: PASS[1, 2], NO\_OPTION*

1231 **2**  
 1232 *M\_GA\_commonC\_result\_decl ( mprotect; sys/mman.h;;;)*  
 1233 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 1234 *Conformance for munmap: PASS[1, 2], NO\_OPTION*

1235 **3**  
 1236 *M\_GA\_macro\_result\_decl(int; mprotect; sys/mman.h;;;)*  
 1237 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 1238 *Conformance for mprotect: PASS, NO\_OPTION*

1239 **4**  
 1240 *M\_GA\_macro\_args ( mprotect; sys/mman.h;;;)*  
 1241 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 1242 *Conformance for mprotect: PASS, NO\_OPTION*

### 12.2.3.2 Description

#### mprotect

1244 **IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
 1245 **SETUP:** Map memory pages using the function *mmap()*.  
 1246 **TEST:** A successful call to the function *mprotect( addr, len, prot)* changes the access  
 1247 protections to be that specified by *prot* for those whole pages containing any part of  
 1248 the address space of the process, starting at address *addr* and continuing for *len* bytes,  
 1249 and returns the value zero.  
 1250 **TR:** Test for PROT\_READ, PROT\_WRITE, and PROT\_NONE individually.  
 1251 **ELSE** *NO\_OPTION*  
 1252 *Conformance for mprotect: PASS,, NO\_OPTION*

1254 **R-1** **IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
 1255 **SETUP:** Map memory pages using the function *mmap()*. Include the header *sys/mman.h*.  
 1256 **TEST:** The constants PROT\_NONE, PROT\_READ, PROT\_WRITE, and PROT\_EXEC  
 1257 are defined and are bitwise distinct.  
 1258 **ELSE** *NO\_OPTION*  
 1259 **SEE:** Assertion *prot\_values* in §12.2.1.2  
 1260 *Conformance for mprotect: PASS,, NO\_OPTION*

1261 **5** **IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
 1262 **SETUP:** Map memory pages using the function *mmap()*.  
 1263 **TEST:** In the call *mprotect( addr, len, prot)*, the only permitted values for *prot* are  
 1264 PROT\_NONE or the bitwise inclusive OR of one or more of the values  
 1265 PROT\_READ, PROT\_WRITE.  
 1266 **ELSE** *NO\_OPTION*  
 1267 *Conformance for mprotect: PASS,, NO\_OPTION*

1268 **R-2** **IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
 1269 **SETUP:** Map memory pages using the function *mmap()*.  
 1270 **TEST:** When an implementation cannot support the combination of access types specified by  
 1271 *prot*, the call to *mprotect()* fails.  
 1272 **ELSE** *NO\_OPTION*  
 1273 **SEE:** Assertion *mprotect\_ENOTSUP* in §12.2.3.4

1274 **D\_1** **IF** *PCTS\_mprotect* and a PCD.1b documents the following **THEN**  
 1275 **TEST:** A PCD.1b that documents whether accesses other than those specified  
 1276 by *prot* does so in §12.2.3.2.

1277           **ELSE NO\_OPTION**  
1278           *Conformance for mprotect: PASS, NO\_OPTION*

1279       **6**       **IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
1280           **SETUP:** Map memory pages using the function *mmap()*.  
1281           **TEST:** All accesses fail where PROT\_WRITE alone has been set.  
1282           **ELSE NO\_OPTION**  
1283           *Conformance for mprotect: PASS,, NO\_OPTION*

1284       **7**       **IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
1285           **SETUP:** Map memory pages using the function *mmap()*.  
1286           **TEST:** All accesses fail where PROT\_NONE alone has been set.  
1287           **ELSE NO\_OPTION**  
1288           *Conformance for mprotect: PASS,, NO\_OPTION*

1289       **8**       **IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
1290           **SETUP:** Map memory pages using the function *mmap()*.  
1291           **TEST:** The implementation supports at least the following values of *prot* PROT\_NONE,  
1292           PROT\_READ, PROT\_WRITE and the inclusive OR of PROT\_READ, and  
1293           PROT\_WRITE.  
1294           **ELSE NO\_OPTION**  
1295           *Conformance for mprotect: PASS,, NO\_OPTION*

1296       **9**       **IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
1297           **SETUP:** Map memory pages using the function *mmap()*.  
1298           **TEST:** When MAP\_PRIVATE was not specified in the original mapping, and  
1299           PROT\_WRITE is specified, mapped objects are opened in the specified address range  
1300           with write permission.  
1301           **TR:** Try with and without closing file descriptors used to map the objects after performing the  
1302           original mapping.  
1303           **ELSE NO\_OPTION**  
1304           *Conformance for mprotect: PASS,, NO\_OPTION*

1305       **D\_2**      **IF** *PCTS\_mprotect* and a PCD.1b documents the following **THEN**  
1306           **TEST:** A PCD.1b that documents the argument *addr* in a call to *mprotect(addr,*  
1307           *len, prot)*, must be a multiple of the page size {PAGESIZE}, does so  
1308           in §12.2.3.2.  
1309           **ELSE NO\_OPTION**  
1310           *Conformance for mprotect: PASS, NO\_OPTION*

1311       **D\_3**      **IF** *PCTS\_mprotect* and a PCD.1b documents the following **THEN**  
1312           **TEST:** A PCD.1b that documents the behavior of this function if the mapping  
1313           was not established by a call to *mmap()*, does so in §12.2.3.2.  
1314           **ELSE NO\_OPTION**  
1315           *Conformance for mprotect: PASS, NO\_OPTION*

1316       **D\_4**      **IF** *PCTS\_mprotect* and a PCD.1b documents the following **THEN**  
1317           **TEST:** A PCD.1b that documents whether or not it supports the *mprotect()*  
1318           function does so in §12.2.3.2.  
1319           **ELSE NO\_OPTION**  
1320           *Conformance for mprotect: PASS, NO\_OPTION*

1321       **12.2.3.3 Returns**

- 1322 **R\_3 IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
 1323       **SETUP:** Map memory pages using the function *mmap()*.  
 1324       **TEST:** When a call to *mprotect()* completes successfully, the interface returns a value of 0.  
 1325       **ELSE NO\_OPTION**  
 1326       **SEE:** Assertion *mprotect* in §12.2.3.2
- 1327 **R\_4 IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
 1328       **SETUP:** Map memory pages using the function *mmap()*.  
 1329       **TEST:** When a call to *mprotect()* completes unsuccessfully, the interface returns a value of  
 1330       -1 and sets *errno* to indicate the error.  
 1331       **ELSE NO\_OPTION**  
 1332       **SEE:** All assertions in §12.2.3.4
- 1333 **10 IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
 1334       **SETUP:** Map memory pages using the function *mmap()*.  
 1335       **TEST:** When *mprotect()* fails and returns [EINVAL], the protections in the pages in the  
 1336       address range starting at *addr* and continuing for *len* bytes are unchanged.  
 1337       **ELSE NO\_OPTION**  
 1338       *Conformance for mprotect: PASS,, NO\_OPTION*
- 1339 **12.2.3.4 Errors**
- 1340 **11 IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
 1341       **SETUP:** Map memory pages using the function *mmap()*.  
 1342       **TEST:** A call to *mprotect()*, when the memory object was not opened for read, regardless of  
 1343       the protection specified, returns a value of -1 and sets *errno* to [EACCESS].  
 1344       **ELSE NO\_OPTION**  
 1345       *Conformance for mprotect: PASS,, NO\_OPTION*
- 1346 **12 IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
 1347       **SETUP:** Map memory pages using the function *mmap()*.  
 1348       **TEST:** A call to *mprotect(addr, len, prot)*, when the memory object was not opened for write,  
 1349       and PROT\_WRITE is specified for a MAP\_SHARED type mapping, returns a value  
 1350       of -1 and sets *errno* to [EACCESS].  
 1351       **ELSE NO\_OPTION**  
 1352       *Conformance for mprotect: PASS,, NO\_OPTION*
- 1353 **13 IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
 1354       **IF** *PCTS\_MAP\_PRIVATE* **THEN**  
 1355           **SETUP:** Map memory pages using the function *mmap()*.  
 1356           **TEST:** A call to *mprotect(addr, len, prot)*, when the *prot* argument  
 1357           specifies PROT\_WRITE on a MAP\_PRIVATE mapping, and there are  
 1358           insufficient memory resources to reserve for locking the private  
 1359           pages, if required, returns a value of -1 and sets *errno* to  
 1360           [EAGAIN].  
 1361           **NOTE:** There is no known reliable test method for this assertion.  
 1362           **ELSE NO\_TEST\_SUPPORT**  
 1363       **ELSE NO\_OPTION**  
 1364       *Conformance for mprotect: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*
- 1365 **14 IF** *PCTS\_mprotect* and *PCTS\_mmap* **THEN**  
 1366       **SETUP:** Map memory pages using the function *mmap()*.  
 1367       **TEST:** A call to *mprotect(addr, len, prot)*, when the addresses in the range starting at *addr*  
 1368       and continuing for *len* bytes are outside the range allowed for the address space of a  
 1369       process returns a value of -1 and sets *errno* to [ENOMEM].

1370 **ELSE NO\_OPTION**  
1371 *Conformance for mprotect: PASS,, NO\_OPTION*

1372 **15 IF PCTS\_mprotect and PCTS\_mmap THEN**  
1373 **SETUP:** Map memory pages using the function *mmap()*.  
1374 **TEST:** A call to *mprotect(addr, len, prot)*, when the addresses in the range starting at *addr*  
1375 and continuing for *len* bytes specify one or more pages that are not mapped, returns  
1376 a value of -1 and sets *errno* to [ENOMEM].  
1377 **ELSE NO\_OPTION**  
1378 *Conformance for mprotect: PASS,, NO\_OPTION*

1379 **16 IF PCTS\_mprotect and PCTS\_mmap THEN**  
1380 **IF PCTS\_MAP\_PRIVATE THEN**  
1381 **SETUP:** Map memory pages using the function *mmap()*.  
1382 **TEST:** A call to *mprotect(addr, len, prot)*, when the *prot* argument specifies  
1383 PROT\_WRITE on a MAP\_PRIVATE mapping, and it would require  
1384 more space than the system is able to supply for locking the private  
1385 pages, if required, returns a value of -1 and sets *errno* to  
1386 [ENOMEM].

1387 **ELSE NO\_TEST\_SUPPORT**  
1388 **ELSE NO\_OPTION**  
1389 *Conformance for mprotect: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1390 **17 IF not PCTS\_mprotect THEN**  
1391 **TEST:** A call to *mprotect()* returns a value of -1 and sets *errno* to [ENOSYS].  
1392 **ELSE NO\_OPTION**  
1393 *Conformance for mprotect: PASS,, NO\_OPTION*

1394 **mprotect\_ENOTSUP**  
1395 **IF PCTS\_mprotect and PCTS\_mmap THEN**  
1396 **SETUP:** Map memory pages using the function *mmap()*.  
1397 **TEST:** A call to *mprotect()*, when the implementation does not support the combination of  
1398 accesses requested in the *prot* argument, returns a value of -1 and sets *errno* to  
1399 [ENOTSUP].  
1400 **ELSE NO\_OPTION**  
1401 *Conformance for mprotect: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1402 **18 IF PCTS\_mprotect and PCTS\_mmap THEN**  
1403 **IF PCTS\_MULTIPLE\_OF\_PAGESIZE**  
1404 and **PCTS\_DETECT\_NOT\_MULTIPLE\_OF\_PAGESIZE THEN**  
1405 **SETUP:** Map memory pages using the function *mmap()*.  
1406 **TEST:** A call to *mprotect(addr, len, prot)*, when the value of *addr* is not  
1407 a multiple of the page size {PAGESIZE}, returns a value of -1 and  
1408 sets *errno* to [EINVAL].  
1409 **ELSE NO\_TEST\_SUPPORT**  
1410 **ELSE NO\_OPTION**  
1411 *Conformance for mprotect: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1412 **12.2.4 Memory Object Synchronization**

1413 **Function: msync()**



1414 **12.2.4.1 Synopsis**1415 **1**1416 *M\_GA\_stdC\_proto\_decl(int; msync; void\*addr, size\_t len; int\_flags; sys/mman.h;;;)*1417 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.31418 *Conformance for msync: PASS[1, 2], NO\_OPTION*1419 **2**1420 *M\_GA\_commonC\_result\_decl ( msync; sys/mman.h;;;)*1421 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.31422 *Conformance for msync: PASS[1, 2], NO\_OPTION*1423 **3**1424 *M\_GA\_macro\_result\_decl(int; msync; sys/mman.h;;;)*1425 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.41426 *Conformance for msync: PASS, NO\_OPTION*1427 **4**1428 *M\_GA\_macro\_args ( msync; sys/mman.h;;;)*1429 **SEE:** Assertion GA\_macro\_args in §2.7.31430 *Conformance for msync: PASS, NO\_OPTION*1431 **12.2.4.2 Description**1432 **msync** **IF** *PCTS\_msync* and *PCTS\_munmap* **THEN**1433 **IF:** *PCTS\_msync\_storage* **THEN**1434 **SETUP:** Map memory pages using the function *mmap()*1435 **TEST:** A successful call to the function *msync()* writes all modified data to permanent  
1436 storage locations in those whole pages containing any part of the address space  
1437 of the process starting at address *addr* and continuing for *len* bytes and returns  
1438 the value zero.1439 **ELSE** *NO\_TEST\_SUPPORT*1440 **ELSE** *NO\_OPTION*1441 *Conformance for msync: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*1442 **5** **IF** *PCTS\_msync* and *PCTS\_munmap* **THEN**1443 **SETUP:** Map memory pages using the function *mmap()*1444 **TEST:** A successful call to the function *msync(addr, len, flags)* where the *flags* argument  
1445 contains the value *MS\_INVALIDATE* invalidates cached copies of the data.1446 **ELSE** *NO\_TEST\_SUPPORT*1447 **ELSE** *NO\_OPTION*1448 *Conformance for msync: PASS, NO\_TEST, NO\_OPTION*1449 **D\_1** **IF** *PCTS\_msync* and a PCD.1b documents the following **THEN**1450 **TEST:** A PCD.1b that documents a call to *msync(addr, len, flags)*, when there  
1451 are no permanent storage locations to write the modified data does so  
1452 in §12.2.4.2.1453 **ELSE** *NO\_OPTION*1454 *Conformance for msync: PASS, NO\_OPTION*1455 **D\_2** **IF** *PCTS\_msync* and a PCD.1b documents the following **THEN**

1456                   **TEST:** A PCD.1b that documents that the argument *addr*, in a call to  
1457                                    *msync(addr, len, flags)*, must be a multiple of the page size  
1458                                    {PAGESIZE}, does so in §12.2.4.2.  
1459                   **ELSE NO\_OPTION**  
1460                   *Conformance for msync: PASS, NO\_OPTION*

1461   **D\_3**   **IF** *PCTS\_msync* and a PCD.1b documents the following **THEN**  
1462                   **TEST:** A PCD.1b that documents whether the implementation also writes out  
1463                                    other file attributes does so in §12.2.4.2.  
1464                   **ELSE NO\_OPTION**  
1465                   *Conformance for msync: PASS, NO\_OPTION*

1466   **6**   **IF** *PCTS\_msync* and *PCTS\_munmap* **THEN**  
1467                    **IF** *PCTS\_msync\_storage* and *PCTS\_MAP\_PRIVATE* **THEN**  
1468                                    **SETUP:** Map memory pages using the function *mmap()*  
1469                                    **TEST:** When the *msync()* function is called on *MAP\_PRIVATE* mapping, any modified  
1470                                    data is not written to the underlying object and does not cause such data to be  
1471                                    made visible to other processes.  
1472                                    **ELSE NO\_TEST\_SUPPORT**  
1473                    **ELSE NO\_OPTION**  
1474                    *Conformance for msync: PASS, NO\_TEST, NO\_OPTION*

1475   **D\_4**   **IF** *PCTS\_msync* and a PCD.1b documents the following **THEN**  
1476                    **TEST:** A PCD.1b that documents whether data in *MAP\_PRIVATE* mappings has  
1477                                    any permanent storage locations does so in §12.2.4.2.  
1478                    **ELSE NO\_OPTION**  
1479                    *Conformance for msync: PASS, NO\_OPTION*

1480   **D\_5**   **IF** *PCTS\_msync* and a PCD.1b documents the following **THEN**  
1481                    **TEST:** A PCD.1b that documents the effect of *msync()* on shared memory  
1482                                    objects §12.2.4.2.  
1483                    **ELSE NO\_OPTION**  
1484                    *Conformance for msync: PASS, NO\_OPTION*

1485   **7**   **IF** *PCTS\_msync* **THEN**  
1486                    **SETUP:** Include the header *sys/msync.h*.  
1487                    **TEST:** The constants *MS\_ASYNC*, *MS\_SYNC*, and *MS\_INVALIDATE* are defined and are bitwise  
1488                                    distinct.  
1489                    **ELSE NO\_OPTION**  
1490                    *Conformance for msync: PASS,, NO\_OPTION*

1491   **8**   **IF** *PCTS\_msync* and *PCTS\_mmap* **THEN**  
1492                    **IF** *PCTS\_msync\_storage* **THEN**  
1493                                    **SETUP:** Map memory pages using the function *mmap()*  
1494                                    **TEST:** When *MS\_ASYNC* the *msync()* returns immediately, once all the write operations  
1495                                    are initiated or queued for servicing.  
1496                                    **ELSE NO\_TEST\_SUPPORT**  
1497                    **ELSE NO\_OPTION**  
1498                    *Conformance for msync: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1499   **9**   **IF** *PCTS\_msync* and *PCTS\_mmap* **THEN**  
1500                    **IF** *PCTS\_msync\_storage* **THEN**  
1501                                    **SETUP:** Map memory pages using the function *mmap()*  
1502                                    **TEST:** When *MS\_SYNC* *msync()* does not return until all write operations are completed  
1503                                    as defined for synchronized I/O data integrity completion.

1504                   **NOTE:** There is no known portable test method for this assertion.  
1505                   **ELSE NO\_TEST\_SUPPORT**  
1506                   **ELSE NO\_OPTION**  
1507                   **SEE:** Assertion GA\_syncIODataIntegrityWrite in §2.2.119  
1508                   *Conformance for msync: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1509   **10**           **IF** *PCTS\_msync* and *PCTS\_mmap* **THEN**  
1510                   **IF** *PCTS\_msync\_storage* **THEN**  
1511                    **SETUP:** Map memory pages using the function *mmap()*  
1512                    **TEST:** When *MS\_ASYNC* is specified, *msync()* all write operations are completed as  
1513                    defined for synchronized I/O data integrity completion.  
1514                    **NOTE:** There is no known portable test method for this assertion.  
1515                    **ELSE NO\_TEST\_SUPPORT**  
1516                    **ELSE NO\_OPTION**  
1517                    **SEE:** Assertion GA\_syncIODataIntegrityWrite in §2.2.119  
1518                    *Conformance for msync: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1519   **R\_1** **IF** *PCTS\_msync* **THEN**  
1520                    **TEST:** A call to *msync()* cannot specify both *MS\_ASYNC* and *MS\_SYNC*  
1521                    **ELSE NO\_OPTION**  
1522                    **SEE:** Assertions *msync\_einval* in §12.2.4.4

1523   **11**           **IF** *PCTS\_msync* and *PCTS\_mmap* **THEN**  
1524                    **IF** *PCTS\_msync\_storage* **THEN**  
1525                    **SETUP:** Map memory pages using the function *mmap()*  
1526                    **TEST:** Following a call to *msync(addr, len, flags)* with *MS\_INVALIDATE* set, references  
1527                    to the object obtain data that was consistent with the permanent storage  
1528                    locations sometime between the call to *msync()* and the first subsequent memory  
1529                    reference to the data.  
1530                    **NOTE:** The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough  
1531                    to write a portable test..  
1532                    **ELSE NO\_TEST\_SUPPORT**  
1533                    **ELSE NO\_OPTION**  
1534                    *Conformance for msync: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1535   **D\_6**           **IF** *PCTS\_msync* and a PCD.1b documents the following **THEN**  
1536                    **TEST:** A PCD.1b that documents the behavior of this function, if the mapping  
1537                    was not established by a call to *mmap()*, does so in §12.2.4.2.  
1538                    **ELSE NO\_OPTION**  
1539                    *Conformance for msync: PASS, NO\_OPTION*

1540   **D\_7**           **IF** *PCTS\_msync* and a PCD.1b documents the following **THEN**  
1541                    **TEST:** A PCD.1b that documents whether or not it supports the *msync()*  
1542                    function does so in §12.2.4.2.  
1543                    **ELSE NO\_OPTION**  
1544                    *Conformance for msync: PASS, NO\_OPTION*

1545   **12.2.4.3 Returns**

1546   **R\_2** **IF** *PCTS\_msync* and *PCTS\_mmap* **THEN**  
1547                    **IF** *PCTS\_msync\_storage* **THEN**  
1548                    **SETUP:** Map memory pages using the function *mmap()*  
1549                    **TEST:** When a call to *msync()* completes successfully, the interface returns a value of  
1550                    0.  
1551                    **NOTE:** There is no known portable test method for this assertion.  
1552                    **ELSE NO\_TEST\_SUPPORT**

1553           **ELSE NO\_OPTION**  
1554           **SEE:**     Assertion msync in §12.2.4.2

1555   **R\_3 IF PCTS\_msync and PCTS\_mmap THEN**  
1556       **IF PCTS\_msync\_storage THEN**  
1557           **SETUP:** Map memory pages using the function *mmap()*  
1558           **TEST:**    When a call to *msync()* completes unsuccessfully, the interface returns a value  
1559                      of -1, and sets *errno* to indicate the error.  
1560           **NOTE:** There is no known portable test method for this assertion.  
1561       **ELSE NO\_TEST\_SUPPORT**  
1562       **ELSE NO\_OPTION**  
1563       **SEE:**     Assertion msync in §12.2.4.4

1564   **12.2.4.4 Errors**

1565   **12 IF PCTS\_msync and PCTS\_mmap THEN**  
1566       **IF PCTS\_msync\_storage THEN**  
1567           **SETUP:** Map memory pages using the function *mmap()*  
1568           **TEST:**    A call to *msync(addr, len, flags)*, when some or all of the address in the range  
1569                      starting at *addr* and continuing for *len* bytes are locked, and *MS\_INVALIDATE* is  
1570                      specified, returns a value of -1 and sets *errno* to [EBUSY].  
1571       **ELSE NO\_TEST\_SUPPORT**  
1572       **ELSE NO\_OPTION**  
1573       Conformance for *msync*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1574   **msync\_einval**  
1575       **IF PCTS\_msync and PCTS\_mmap THEN**  
1576           **IF PCTS\_msync\_storage THEN**  
1577           **SETUP:** Map memory pages using the function *mmap()*  
1578           **TEST:**    A call to *msync(addr, len, flags)*, when the value in *flags* is invalid, returns a  
1579                      value of -1 and sets *errno* to [EINVAL].  
1580           **TR:** Test with *flags* having both *MS\_ASYNC* and *MS\_SYNC* in it.  
1581           **NOTE:**    A subroutine is recommended that either returns a *flags* argument that includes  
1582                      unimplemented flags or indicates that there is no way to generate a *flags*  
1583                      argument that includes unimplemented flags on the system.  
1584       **ELSE NO\_TEST\_SUPPORT**  
1585       **ELSE NO\_OPTION**  
1586       Conformance for *msync*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1587   **13 IF PCTS\_msync and PCTS\_mmap THEN**  
1588       **IF PCTS\_msync\_storage THEN**  
1589           **SETUP:** Map memory pages using the function *mmap()*  
1590           **TEST:**    A call to *msync(addr, len, flags)*, when the addresses in the range starting at *add*  
1591                      and continuing for *len* bytes are outside the range allowed for the address space  
1592                      of a process, returns a value of -1 and sets *errno* to [ENOMEM].  
1593       **ELSE NO\_TEST\_SUPPORT**  
1594       **ELSE NO\_OPTION**  
1595       Conformance for *msync*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1596   **14 IF PCTS\_msync and PCTS\_mmap THEN**  
1597       **IF PCTS\_msync\_storage THEN**  
1598           **SETUP:** Map memory pages using the function *mmap()*  
1599           **TEST:**    A call to *msync(addr, len, flags)*, when the address in the range starting at *addr*  
1600                      and continuing for *len* specify one of more pages that are not mapped, returns  
1601                      a value of -1 and sets *errno* to [ENOMEM].  
1602           **TR:** Test for one page not mapped and for more than one page not mapped.  
1603       **ELSE NO\_TEST\_SUPPORT**  
1604       **ELSE NO\_OPTION**  
1605       Conformance for *msync*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1606 **15** **IF** not *PCTS\_msync* **THEN**  
 1607 **TEST:** A call to *msync()*, returns a value of -1 and sets *errno* to [ENOSYS].  
 1608 **ELSE** *NO\_OPTION*  
 1609 *Conformance for msync: PASS, NO\_OPTION*

1610 **16** **IF** *PCTS\_msync* and *PCTS\_mmap* **THEN**  
 1611 **IF** *PCTS\_msync\_storage* and *PCTS\_MULTIPLE\_OF\_PAGESIZE* and  
 1612 *PCTS\_DETECT\_NOT\_MULTIPLE\_OF\_PAGESIZE* **THEN**  
 1613 **SETUP:** Map memory pages using the function *mmap()*  
 1614 **TEST:** A call to *msync(addr, len, flags)*, when the value of *addr* is not a multiple of the  
 1615 page size {*PAGESIZE*}, returns a value of -1 and sets *errno* to [EINVAL].  
 1616 **ELSE** *NO\_TEST\_SUPPORT*  
 1617 **ELSE** *NO\_OPTION*  
 1618 *Conformance for msync: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

## 1619 **12.3 Shared Memory Functions**

### 1620 **12.3.1 Open a Shared Memory Object**

1621 Function: *shm\_open()*

#### 1622 **12.3.1.1 Synopsis**

1623 **1**  
 1624 *M\_GA\_stdC\_proto\_decl( int; shm\_open; const char\*name, int oflag, mod\_t mode;*  
 1625 *sys/mman.h;;;)*  
 1626 **SEE:** Assertion *GA\_stdC\_proto\_decl* in §2.7.3  
 1627 *Conformance for shm\_open: PASS[1, 2], NO\_OPTION*

1628 **2**  
 1629 *M\_GA\_commonC\_result\_decl ( shm\_open; sys/mman.h;;;)*  
 1630 **SEE:** Assertion *GA\_commonC\_int\_result\_decl* in §2.7.3  
 1631 *Conformance for shm\_open.: PASS[1, 2], NO\_OPTION*

1632 **3**  
 1633 *M\_GA\_macro\_result\_decl(int; shm\_open; sys/mman.h;;;)*  
 1634 **SEE:** Assertion *GA\_macro\_result\_decl* in §1.3.4  
 1635 *Conformance for shm\_open: PASS, NO\_OPTION*

1636 **4**  
 1637 *M\_GA\_macro\_args ( shm\_open; sys/mman.h;;;)*  
 1638 **SEE:** Assertion *GA\_macro\_args* in §2.7.3  
 1639 *Conformance for shm\_open: PASS, NO\_OPTION*

#### 1640 **12.3.1.2 Description**

1641 **shm\_open**  
 1642 **IF** *PCTS\_shm\_open* **THEN**  
 1643 **TEST:** A successful call to the *shm\_open()* function creates an open file description that  
 1644 refers to the shared memory object, and returns a valid file descriptor that refers to  
 1645 that open file description.  
 1646 **ELSE** *NO\_OPTION*  
 1647 *Conformance for shm\_open: PASS, NO\_OPTION*

1648 **5** **IF** *PCTS\_shm\_open* **THEN**

1649                   **TEST:**     The file descriptor returned by *shm\_open()* is a nonnegative integer.  
1650                   **ELSE NO\_OPTION**  
1651                   *Conformance for shm\_open: PASS, NO\_OPTION*

1652   **D\_1**   **IF** *PCTS\_shm\_open* and a PCD.1b documents the following **THEN**  
1653                   **TEST:**     A PCD.1b that documents whether the shared memory object *name*  
1654                                 appears in the file system and is visible to other functions that take  
1655                                 pathnames as arguments does so in §12.3.1.2.  
1656                   **ELSE NO\_OPTION**  
1657                   *Conformance for shm\_open: PASS, NO\_OPTION*

1658   **6**  
1659                   ***M\_GA\_portableFileNames( shm\_open)***  
1660                   **SEE:**     Assertion *GA\_portableFileNames* in §2.2.4.0  
1661                   *Conformance for shm\_open: PASS, NO\_OPTION*

1662   **7**  
1663                   ***M\_GA\_upperLowerNames ( shm\_open)***  
1664                   **SEE:**     Assertion *GA\_upperLowerNames* in §2.2.2.40  
1665                   *Conformance for shm\_open:: PASS, NO\_OPTION*

1666   **8**  
1667                   ***M\_GA\_PRNoTRUNC( shm\_open)***  
1668                   **SEE:**     Assertion *GA\_PRNoTrunc* in §2.3.6  
1669                   *Conformance for shm\_open: PASS, NO\_OPTION*

1670   **9**  
1671                   ***M\_GA\_PRNoTruncError ( shm\_open)***  
1672                   **SEE:**     Assertion *GA\_macro\_args* in §2.7.3  
1673                   *Conformance for shm\_open: PASS, NO\_OPTION*

1674   **10**   **IF** *PCTS\_shm\_open* **THEN**  
1675                   **TEST:**     When *name* begins with the slash character, then processes calling *shm\_open(name,*  
1676                                 *oflag, mode)*, with the same value of *name* refer to the same shared memory object,  
1677                                 as long as that name has not been removed.  
1678                   **TR:**     Test using two different processes.  
1679                   **ELSE NO\_OPTION**  
1680                   *Conformance for shm\_open: PASS,, NO\_OPTION*

1681   **D\_2**   **IF** *PCTS\_shm\_open* **THEN**  
1682                   **TEST:**     A PCD.1b that documents the effect if the *name* to *shm\_open(name,*  
1683                                 *oflag, mode)* does not begin with the slash character in §12.3.1.2.  
1684                   **ELSE NO\_OPTION**  
1685                   *Conformance for shm\_open: PASS, NO\_OPTION*

1686   **D\_3**   **IF** *PCTS\_shm\_open* **THEN**  
1687                   **TEST:**     A PCD.1b that documents the interpretation of slash characters other  
1688                                 than the leading slash character in the *name* argument to  
1689                                 *shm\_open(name, oflag, mode)* in §12.3.1.2.  
1690                   **ELSE NO\_OPTION**  
1691                   *Conformance for shm\_open: PASS, NO\_OPTION*

- 1692 **11** **IF** *PCTS\_shm\_open* **THEN**  
 1693 **TEST:** A successful call to *shm\_open()*, returns a file descriptor for the shared memory object  
 1694 that is the lowest-numbered file descriptor not currently open for that process..  
 1695 **ELSE NO\_OPTION**  
 1696 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1697 **12** **IF** *PCTS\_shm\_open* **THEN**  
 1698 **TEST:** The process does not share the open file description created by *shm\_open()* with any  
 1699 other processes.  
 1700 **ELSE NO\_OPTION**  
 1701 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1702 **D\_4** **IF** *PCTS\_shm\_open* and a PCD.1b documents the following **THEN**  
 1703 **TEST:** A PCD.1b that documents whether the file offset is set by  
 1704 *shm-open()* does so in §12.3.1.2.  
 1705 **ELSE NO\_OPTION**  
 1706 *Conformance for shm\_open: PASS, NO\_OPTION*
- 1707 **13** **IF** *PCTS\_shm\_open* **THEN**  
 1708 **TEST:** The FD\_CLOEXEC file descriptor flag associated with the new file descriptor is set by  
 1709 *shm\_open()*  
 1710 **ELSE NO\_OPTION**  
 1711 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1712 **14** **IF** *PCTS\_shm\_open* **THEN**  
 1713 **TEST:** Following a call to *shm\_open(name, oflag, mode)*, the file status flags and file access  
 1714 modes of the open file description are set according to the value of *oflag*.  
 1715 **ELSE NO\_OPTION**  
 1716 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1717 **15** **IF** *PCTS\_shm\_open* **THEN**  
 1718 **SETUP:** Include the header `<sys/mman.h>`  
 1719 **TEST:** The constants `O_RDONLY`, `O_RDWR`, `O_CREAT`, `O_EXCL` and `O_TRUNC` are defined, and  
 1720 have the same values as defined in the header `<fcntl.h>`  
 1721 **ELSE NO\_OPTION**  
 1722 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1723 **16** **IF** *PCTS\_shm\_open* **THEN**  
 1724 **SETUP:** Include the header `<sys/mman.h>`  
 1725 **TEST** In a call to *shm\_open(name, oflag, mode)*, the *oflag* argument must be the bitwise  
 1726 inclusive OR of one or more of the following flags: `O_RDONLY`, `O_RDWR`, `O_CREAT`,  
 1727 `O_EXCL` and `O_TRUNC`.  
 1728 **ELSE NO\_OPTION**  
 1729 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1730 **17** **IF** *PCTS\_shm\_open* **THEN**  
 1731 **TEST:** In the call to *shm\_open(name, oflag, mode)*, *oflag* must have either `O_RDONLY`, or  
 1732 `O_RDWR` set.  
 1733 **ELSE NO\_OPTION**  
 1734 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1735 **18** **IF** *PCTS\_shm\_open* **THEN**  
 1736 **TEST:** In the call to *shm\_open(name, oflag, mode)*, *oflag* must have both `O_RDONLY`, or  
 1737 `O_RDWR` set.  
 1738 **ELSE NO\_OPTION**  
 1739 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1740 **19** **IF** *PCTS\_shm\_open* **THEN**

- 1741                   **TEST:** In the call to *shm\_open(name, oflag, mode)*, any combination of the remaining flags -  
1742   O\_CREAT, O\_EXCL, and O\_TRUNC may be specified in *oflag*.  
1743                   **ELSE NO\_OPTION**  
1744                   Conformance for *shm\_open*: PASS,, NO\_OPTION
- 1745    **20**       **IF PCTS\_shm\_open THEN**  
1746                   **TEST:** The flag O\_CREAT has no effect in a call to *shm\_open(name, oflag, mode)* that refers  
1747   to a shared-memory object that was previously created with the O\_EXCL flag unset.  
1748                   **ELSE NO\_OPTION**  
1749                   Conformance for *shm\_open*: PASS,, NO\_OPTION
- 1750    **21**       **IF PCTS\_shm\_open THEN**  
1751                   **TEST:** A call to *shm\_open(name, oflag, mode)* creates a shared-memory object when the flag  
1752   O\_CREAT is set and the shared memory object did not exist previously  
1753                   **ELSE NO\_OPTION**  
1754                   Conformance for *shm\_open*: PASS,, NO\_OPTION
- 1755    **22**       **IF PCTS\_shm\_open THEN**  
1756                   **SETUP:** Create a shared-memory object by calling *shm\_open (name, oflag, mode)* with the  
1757   *oflag* argument set to O\_CREAT  
1758                   **TEST:** The user ID of newly created, shared-memory object is set to the effective user ID of  
1759   the process.  
1760                   **ELSE NO\_OPTION**  
1761                   Conformance for *shm\_open*: PASS,, NO\_OPTION
- 1762    **23**       **IF PCTS\_shm\_open THEN**  
1763                   **SETUP:** Create a shared-memory object by calling *shm\_open (name, oflag, mode)* with the  
1764   *oflag* argument set to O\_CREAT  
1765                   **TEST:** The group ID of newly created, shared-memory object is set to the effective user ID of  
1766   the process.  
1767                   **ELSE NO\_OPTION**  
1768                   Conformance for *shm\_open*: PASS,, NO\_OPTION
- 1769    **24**       **IF PCTS\_shm\_open THEN**  
1770                   **TEST:** When a shared-memory object is created by calling *shm\_open(name, oflag, mode)*  
1771   with the *oflag* argument set to O\_CREAT, the permission bits are set to the value of the  
1772   *mode* argument, except for those set in the file mode creation mask of the process.  
1773                   **ELSE NO\_OPTION**  
1774                   Conformance for *shm\_open*: PASS,, NO\_OPTION
- 1775    **D\_5**       **IF PCTS\_shm\_open** and a PCD.1b documents the following **THEN**  
1776                   **TEST:** A PCD.1b that documents the effect when bits in *mode* other than the  
1777   file permission bits are set during creation of a shared-memory object  
1778   by a call to *shm\_open(name, oflag, mode)* does so in §12.3.1.2.  
1779                   **ELSE NO\_OPTION**  
1780                   Conformance for *shm\_open*: PASS, NO\_OPTION
- 1781    **25**       **IF PCTS\_shm\_open THEN**  
1782                   **TEST:** In the call *shm\_open(name, oflag, mode)* the *mode* argument does not affect whether  
1783   the shared memory object is opened for reading, for writing, or for both.  
1784                   **TR:** Try all three, with all possibly mode bit combinations.  
1785                   **ELSE NO\_OPTION**  
1786                   Conformance for *shm\_open*: PASS,, NO\_OPTION
- 1787    **26**       **IF PCTS\_shm\_open THEN**  
1788                   **TEST:** A newly created shared memory object has a size of zero  
1789                   **ELSE NO\_OPTION**  
1790                   Conformance for *shm\_open*: PASS,, NO\_OPTION



- 1791 **R\_1 IF PCTS\_shm\_open THEN**  
 1792     **TEST:** When O\_EXCL and O\_CREAT are set, and the shared memory object already exists,  
 1793                *shm\_open()* fails.  
 1794     **ELSE NO\_OPTION**  
 1795     **SEE:** Assertion *shm\_exist\_err* in §12.3.1.4
- 1796 **27 IF PCTS\_shm\_open THEN**  
 1797     **TEST:** The check by *shm\_open()* for the existence of the shared memory object and the  
 1798                creation of the object if it does not exist are atomic, with respect to other processes  
 1799                executing *shm\_open()* with O\_EXCL and O\_CREAT set, and naming the same shared  
 1800                memory object.  
 1801     **NOTE:** There is no known reliable test method for this assertion.  
 1802     **ELSE NO\_OPTION**  
 1803     *Conformance for shm\_open: PASS, NO\_TEST, NO\_OPTION*
- 1804 **D\_6 IF PCTS\_shm\_open and a PCD.1b documents the following THEN**  
 1805     **TEST:** A PCD.1b that documents the result of a call to *shm\_open(name, oflag,*  
 1806                *mode)* if O\_EXCL is set and O\_CREAT is not set does so in §12.3.1.2.  
 1807     **ELSE NO\_OPTION**  
 1808     *Conformance for shm\_open: PASS, NO\_OPTION*
- 1809 **28 IF PCTS\_shm\_open THEN**  
 1810     **TEST:** When a shared-memory object exists, a successful call to *shm\_open(name, oflag,*  
 1811                *mode)* specifying both O\_RDWR and O\_TRUNC, truncates the object to zero length.  
 1812     **ELSE NO\_OPTION**  
 1813     *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1814 **29 IF PCTS\_shm\_open THEN**  
 1815     **TEST:** When a shared-memory object exists, a successful call to *shm\_open(name, oflag,*  
 1816                *mode)* specifying both O\_RDWR and O\_TRUNC, leaves the mode and owner unchanged.  
 1817     **ELSE NO\_OPTION**  
 1818     *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1819 **D\_7 IF PCTS\_shm\_open and a PCD.1b documents the following THEN**  
 1820     **TEST:** A PCD.1b that documents the result of specifying both O\_TRUNC and  
 1821                O\_RDONLY when calling *shm\_open(name, oflag, mode)* does so in  
 1822                §12.3.1.2.  
 1823     **ELSE NO\_OPTION**  
 1824     *Conformance for shm\_open: PASS, NO\_OPTION*
- 1825 **30 IF PCTS\_shm\_open THEN**  
 1826     **TEST:** When a shared-memory object is created, by *shm\_open( )*, the state of the shared  
 1827                memory object, including all data associated with the shared memory object, persists  
 1828                until the shared memory object is unlinked and all other references are gone.  
 1829     **ELSE NO\_OPTION**  
 1830     *Conformance for shm\_open: PASS, NO\_OPTION*
- 1831 **D\_8 IF PCTS\_shm\_open and a PCD.1b documents the following THEN**  
 1832     **TEST:** A PCD.1b that documents whether the name and state of a shared memory object  
 1833                remain valid after a system reboot does so in §12.3.1.2.  
 1834     **ELSE NO\_OPTION**  
 1835     *Conformance for shm\_open: PASS, NO\_OPTION*
- 1836 **D\_9 IF PCTS\_shm\_open and a PCD.1b documents the following THEN**  
 1837     **TEST:** A PCD.1b that documents whether or not it supports the *shm\_open()*  
 1838                function does so in §12.3.1.2.

1839                   **ELSE NO\_OPTION**  
 1840                   *Conformance for shm\_open: PASS, NO\_OPTION*

1841    **12.3.1.3 Returns**

1842    **R\_2 IF PCTS\_shm\_open THEN**  
 1843            **TEST:**    When a call to *shm\_open()* completes successfully, the interface returns a nonnegative  
 1844                           integer representing the lowest numbered unused file descriptor  
 1845            **ELSE NO\_OPTION**  
 1846            **SEE:**      Assertion *shm\_open* in §12.3.1.2

1847    **R\_3 IF PCTS\_shm\_open THEN**  
 1848            **TEST:**    When a call to *shm\_open()* completes unsuccessfully, the interface returns a value of  
 1849                           -1, and sets *errno* to indicate the error  
 1850            **ELSE NO\_OPTION**  
 1851            **SEE:**      All assertions in §12.3.1.4

1852    **12.3.1.4 Errors**

1853    **31 IF PCTS\_shm\_open THEN**  
 1854            **TEST:**    A call to *shm\_open( )*, when the shared memory object exists and the permission  
 1855                           specified by *oflag* are denied, returns a value of -1 and sets *errno* to [EACCESS].  
 1856            **ELSE NO\_OPTION**  
 1857            *Conformance for shm\_open: PASS,, NO\_OPTION*

1858    **32 IF PCTS\_shm\_open THEN**  
 1859            **TEST:**    A call to *shm\_open( )*, when the shared memory object does not exist and permission  
 1860                           to create the shared memory object is denied, returns a value of -1 and sets *errno* to  
 1861                           [EACCESS].  
 1862            **ELSE NO\_OPTION**  
 1863            *Conformance for shm\_open: PASS,, NO\_OPTION*

1864    **33 IF PCTS\_shm\_open THEN**  
 1865            **TEST:**    A call to *shm\_open( )*, when O\_TRUNC is specified and write permission is denied,  
 1866                           returns a value of -1 and sets *errno* to [EACCESS].  
 1867            **ELSE NO\_OPTION**  
 1868            *Conformance for shm\_open: PASS,, NO\_OPTION*

1869    **shm\_exist\_err**  
 1870            **IF PCTS\_shm\_open THEN**  
 1871                    **TEST:**    A call to *shm\_open( )*, when O\_CREAT and O\_EXCL are set and the named shared  
 1872                           memory object already exists, returns a value of -1 and sets *errno* to [EEXIST].  
 1873            **ELSE NO\_OPTION**  
 1874            *Conformance for shm\_open: PASS,, NO\_OPTION*

1875    **34 IF PCTS\_shm\_open THEN**  
 1876            **TEST:**    A call to *shm\_open( )*, when the *shm\_open()* operation is interrupted by a signal,  
 1877                           returns a value of -1 and sets *errno* to [EINTR].  
 1878            **ELSE NO\_OPTION**  
 1879            *Conformance for shm\_open: PASS,, NO\_OPTION*

1880    **35 IF PCTS\_shm\_open THEN**  
 1881            **TEST:**    A call to *shm\_open( )*, when the *shm\_open()* operation is not supported for the given  
 1882                           name, returns a value of -1 and sets *errno* to [EINVAL].

- 1883                   **NOTE:** A subroutine is recommended that either returns a name for which *shm\_open* is not  
 1884 supported, or indicates that there is no way to generate a name for which *shm\_open()*  
 1885 is not supported, on the system.  
 1886 **ELSE NO\_OPTION**  
 1887 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1888 **D\_10 IF PCTS\_shm\_open THEN**  
 1889           **TEST:** A PCD.1b that documents under what circumstances [EINVAL] may  
 1890 be returned in §12.3.1.2.  
 1891 **ELSE NO\_OPTION**  
 1892 *Conformance for shm\_open: PASS, NO\_OPTION*
- 1893 **36 IF PCTS\_shm\_open THEN**  
 1894           **TEST:** A call to *shm\_open( )*, when too many file descriptors are currently in use by this  
 1895 process, returns a value of -1 and sets *errno* to [EMFILE].  
 1896 **ELSE NO\_OPTION**  
 1897 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1898 **37 IF PCTS\_shm\_open THEN**  
 1899           **IF {PATH\_MAX} <= PCTS\_PATH\_MAX THEN**  
 1900           **TEST:** A call to *shm\_open( )*, when the length of the *name* string exceeds {PATH\_MAX}  
 1901 returns a value of -1 and sets *errno* to [ENAMETOOLONG].  
 1902           **ELSE NO\_TEST\_SUPPORT**  
 1903 **ELSE NO\_OPTION**  
 1904 *Conformance for shm\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 1905 **38 IF PCTS\_shm\_open and { \_POSIX\_NO\_TRUNC } THEN**  
 1906           **IF {NAME\_MAX} <= PCTS\_NAME\_MAX THEN**  
 1907           **TEST:** A call to *shm\_open( )*, when a pathname component is longer than {NAME\_MAX}  
 1908 returns a value of -1 and sets *errno* to [ENAMETOOLONG].  
 1909           **ELSE NO\_TEST\_SUPPORT**  
 1910 **ELSE NO\_OPTION**  
 1911 *Conformance for shm\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*
- 1912 **39 IF PCTS\_shm\_open THEN**  
 1913           **TEST:** A call to *shm\_open( )*, when too many shared memory objects are currently open in the  
 1914 system, returns a value of -1 and sets *errno* to [EINFILE].  
 1915 **ELSE NO\_OPTION**  
 1916 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1917 **40 IF PCTS\_shm\_open THEN**  
 1918           **TEST:** A call to *shm\_open( )*, when O\_CREAT is not set and the named shared memory object  
 1919 does not exist, returns a value of -1 and sets *errno* to [ENOENT].  
 1920 **ELSE NO\_OPTION**  
 1921 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1922 **41 IF PCTS\_shm\_open THEN**  
 1923           **TEST:** A call to *shm\_open( )*, when there is insufficient space for the creation of the new  
 1924 shared memory object, returns a value of -1 and sets *errno* to [ENOSPC].  
 1925           **NOTE:** There is no known reliable test method for this assertion.  
 1926 **ELSE NO\_OPTION**  
 1927 *Conformance for shm\_open: PASS,, NO\_OPTION*
- 1928 **42 IF not PCTS\_shm\_open THEN**  
 1929           **TEST:** A call to *shm\_open( )*, returns a value of -1 and sets *errno* to [ENOSYS].  
 1930 **ELSE NO\_OPTION**  
 1931 *Conformance for shm\_open: PASS,, NO\_OPTION*

1932 **12.3.2 Remove a Shared Memory Object**1933 Function: *shm\_unlink()*1934 **12.3.2.1 Synopsis**1935 **1**1936 *M\_GA\_stdC\_proto\_decl( int; shm\_unlink; const char\*name,;;;)*1937 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.31938 *Conformance for shm\_unlink: PASS[1, 2], NO\_OPTION*1939 **2**1940 *M\_GA\_commonC\_result\_decl ( shm\_unlink;;;)*1941 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.31942 *Conformance for shm\_unlink:: PASS[1, 2], NO\_OPTION*1943 **3**1944 *M\_GA\_macro\_result\_decl(int; shm\_unlink;;;)*1945 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.41946 *Conformance for shm\_open: PASS, NO\_OPTION*1947 **4**1948 *M\_GA\_macro\_args ( shm\_unlink;;;)*1949 **SEE:** Assertion GA\_macro\_args in §2.7.31950 *Conformance for shm\_open: PASS, NO\_OPTION*1951 **12.3.2.2 Description**1952 **shm\_unlink**1953 **IF** *PCTS\_shm\_unlink* **THEN**1954 **TEST:** A successful call to the function *shm\_unlink()* removes the name of the shared  
1955 memory object named by the string pointed to by *name*, and returns zero.1956 **ELSE** *NO\_OPTION*1957 *Conformance for shm\_unlink: PASS,, NO\_OPTION*1958 **5**1958 **IF** *PCTS\_shm\_unlink* **THEN**1959 **IF** *PCTS\_shm\_open* **THEN**1960 **TEST:** When one or more references to the shared memory object exist when the object  
1961 is unlinked, the removal of the memory object contents is postponed until all  
1962 open references to the shared memory object have been removed.1963 **ELSE** *NO\_TEST\_SUPPORT*1964 **ELSE** *NO\_OPTION*1965 *Conformance for shm\_unlink: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*1966 **6**1966 **IF** *PCTS\_shm\_unlink* **THEN**1967 **IF** *PCTS\_shm\_open* and *PCTS\_mmap* **THEN**1968 **TEST:** When one or more references to the shared memory object exist when the object  
1969 is unlinked, the removal of the memory object contents is postponed until all  
1970 map references to the shared memory object have been removed.1971 **ELSE** *NO\_TEST\_SUPPORT*1972 **ELSE** *NO\_OPTION*1973 *Conformance for shm\_unlink: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*1974 **D\_1** **IF** *PCTS\_shm\_unlink* and a PCD.1b documents the following **THEN**

1975                   **TEST:** A PCD.1b that documents whether or not it supports the *shm\_unlink()*  
 1976                   function does so in §12.3.2.2.  
 1977                   **ELSE NO\_OPTION**  
 1978                   *Conformance for shm\_unlink: PASS, NO\_OPTION*

1979           **R\_1 IF PCTS\_shm\_unlink THEN**  
 1980                   **TEST:** When a call to, *shm\_unlink()* completes successfully, the interface returns a value of  
 1981                   0.  
 1982                   **ELSE NO\_OPTION**  
 1983                   **SEE:** All assertions in §12.3.2.2

1984           **R\_2 IF PCTS\_shm\_unlink THEN**  
 1985                   **TEST:** When a call to, *shm\_unlink()* completes unsuccessfully, the interface returns a value  
 1986                   of -1, and sets *errno* to indicate the error, and the named shared memory object is  
 1987                   unchanged.  
 1988                   **ELSE NO\_OPTION**  
 1989                   **SEE:** All assertions in §12.3.2.4

1990           **12.3.2.4 Errors**

1991           **7 IF PCTS\_shm\_unlink and {\_POSIX\_NO\_TRUNC} THEN**  
 1992                   **TEST:** A call to *shm\_unlink( )*, when permission is denied to unlink the named shared  
 1993                   memory object, returns a value of -1 and sets *errno* to [EACCESS].  
 1994                   **ELSE NO\_OPTION**  
 1995                   *Conformance for shm\_unlink: PASS, NO\_OPTION*

1996           **8 IF PCTS\_shm\_unlink THEN**  
 1997                   **IF {POSIX\_NO\_TRUNC} and {NAME\_MAX} <= PCTS\_NAME\_MAX THEN**  
 1998                   **TEST:** A call to *shm\_unlink( )*, when the length of the *name* string exceeds  
 1999                   {NAME\_MAX} returns a value of -1 and sets *errno* to [ENAMETOOLONG].  
 2000                   **ELSE NO\_TEST\_SUPPORT**  
 2001                   **ELSE NO\_OPTION**  
 2002                   *Conformance for shm\_unlink: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

2003           **9 IF PCTS\_shm\_unlink THEN**  
 2004                   **TEST:** A call to *shm\_unlink(name )*, when the named shared memory object does not exist,  
 2005                   returns a value of -1 and sets *errno* to [ENOENT].  
 2006                   **ELSE NO\_OPTION**  
 2007                   *Conformance for shm\_unlink: PASS, NO\_OPTION*

2008           **10 IF not PCTS\_shm\_unlink THEN**  
 2009                   **TEST:** A call to *shm\_unlink( )*, returns a value of -1 and sets *errno* to [ENOSYS].  
 2010                   **ELSE NO\_OPTION**  
 2011                   *Conformance for shm\_unlink: PASS, NO\_OPTION*

This page is intentionally blank.

## Section 13: Execution Scheduling

### 180 13.1 Scheduling Parameters

181 **1**       **SETUP:** Include the header `<sched.h>`.  
 182       **TEST:**    The structure `sched_param` is defined, and has the member

183

184	Member	Member	
185	Type	Name	Description

186	<code>int</code>	<code>sched_priority</code>	Process execution scheduling priority.
-----	------------------	-----------------------------	--

187           *Conformance for sched\_param: PASS*

188 **D\_1** **IF** a PCD.1b documents the following **THEN**

189       **TEST:**    A PCD.1b that documents extensions to `sched_param`, as permitted in POSIX.1b{3}  
 190                   §1.3.1.1 item (2), does so in §13.1.

191       **ELSE** *NO\_OPTION*

192       *Conformance for sched\_param: PASS, NO\_OPTION*

193 **2**       **SETUP:** Include the header `<sched.h>`.

194       **TEST:**    Extensions to `sched_param` that may change the behavior of the application with respect to  
 195                   this standard when those fields in the structure are uninitialized, are enabled as required by  
 196                   POSIX.1b {3} §1.3.1.1.

197       **NOTE:**    The corresponding statement in IEEE Std 1003.1b-1993        is not specific enough to write  
 198                   a portable test.

199       *Conformance for sched\_param: PASS, NO\_TEST*

200 **3**       **SETUP:** Include the header `<sched.h>`.

201       **TEST:**    The symbols allowed by this standard to be in the header `<time.h>` are visible.

202       *Conformance for sched\_param: PASS*

### 203 13.2 Scheduling Policies

204 **1**       **TEST:**    The implementation makes the process at the head of the highest priority nonempty process  
 205                   list a running process, regardless of its associated policy.

206       *Conformance for sched\_policy: PASS*

207 **2**       **TEST:**    A running process is then removed from its process list.

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

209       *Conformance for sched\_policy: PASS, NO\_TEST*

210 **D\_1** **IF** a PCD.1b documents the following **THEN**  
 211       **TEST:** A PCD.1b that documents scheduling policies other SCHED\_FIFO SCHED\_RR  
 212               SCHED\_OTHER, does so in §13.2.  
 213       **ELSE NO\_OPTION**  
 214       *Conformance for sched\_policy: PASS, NO\_OPTION*

215 **3**       **SETUP:** Include the header <sched.h> .  
 216       **TEST:** The constants SCHED\_FIFO SCHED\_RR SCHED\_OTHER are defined and are bitwise distinct.  
 217       *Conformance for sched\_policy: PASS*

218 **13.2.1 SCHED\_FIFO**

219 **sched\_fifo1**  
 220       **IF** PCTS\_sched\_setscheduler **THEN**  
 221        **IF** PCTS\_GAP\_sched\_setscheduler **THEN**  
 222         **TEST:** Processes scheduled under the FIFO scheduling policy are chosen from a process  
 223               list ordered by the time its processes have been on the list without being  
 224               executed.  
 225        **ELSE NO\_TEST\_SUPPORT**  
 226       **ELSE NO\_OPTION**  
 227       *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

228 **sched\_fifo2**  
 229       **IF** PCTS\_sched\_setscheduler **THEN**  
 230        **IF** PCTS\_GAP\_sched\_setscheduler **THEN**  
 231         **TEST:** Under the FIFO scheduling policy, when a running process becomes a preempted  
 232               process, it becomes the head of the process list for its priority.  
 233         **NOTE:** There is no known portable test method for this assertion.  
 234        **ELSE NO\_TEST\_SUPPORT**  
 235       **ELSE NO\_OPTION**  
 236       *Conformance for sched\_policy: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

237 **sched\_fifo3**  
 238       **IF** PCTS\_sched\_setscheduler **THEN**  
 239        **IF** PCTS\_GAP\_sched\_setscheduler **THEN**  
 240         **TEST:** Under the SCHED\_FIFO policy, when a blocked process becomes a runnable  
 241               process, it becomes the tail of the process list for its priority.  
 242        **ELSE NO\_TEST\_SUPPORT**  
 243       **ELSE NO\_OPTION**  
 244       *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

245 **sched\_fifo4**  
 246       **IF** PCTS\_sched\_setscheduler **THEN**  
 247        **IF** PCTS\_GAP\_sched\_setscheduler **THEN**  
 248         **TEST:** Under the SCHED\_FIFO policy, when a running process calls the  
 249               sched\_setscheduler() function, the process specified in the function call is  
 250               modified to the specified policy and the priority specified by the *param*  
 251               argument.  
 252        **ELSE NO\_TEST\_SUPPORT**  
 253       **ELSE NO\_OPTION**  
 254       *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

255 **sched\_fifo5**  
 256       **IF** PCTS\_sched\_setscheduler **THEN**  
 257        **IF** PCTS\_GAP\_sched\_setscheduler **THEN**  
 258         **TEST:** Under the SCHED\_FIFO policy, if the process whose policy and priority has been  
 259               modified is a running process or is runnable, it then becomes the tail of the  
 260               process list for its new priority.



261                   **ELSE NO\_TEST\_SUPPORT**  
 262                   **ELSE NO\_OPTION**  
 263                   *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

264 **sched\_fifo6**  
 265           **IF PCTS\_sched\_setscheduler and PCTS\_sched\_setparam THEN**  
 266               **IF PCTS\_GAP\_sched\_setscheduler and PCTS\_GAP\_sched\_setparam THEN**  
 267                   **TEST:** Under the SCHED\_FIFO policy, when a running process calls the  
 268                               *sched\_setparam()* function, the priority of the process specified in the function  
 269                               call is modified to the priority specified by the *param* argument.  
 270                   **ELSE NO\_TEST\_SUPPORT**  
 271                   **ELSE NO\_OPTION**  
 272                   *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

273 **sched\_fifo7**  
 274           **IF PCTS\_sched\_setscheduler and PCTS\_sched\_setparam THEN**  
 275               **IF PCTS\_GAP\_sched\_setscheduler and PCTS\_GAP\_sched\_setparam THEN**  
 276                   **TEST:** Under the SCHED\_FIFO policy, if a process whose priority has been modified is  
 277                               a running process or is runnable, it then becomes the tail of the process list for  
 278                               its new priority.  
 279                   **ELSE NO\_TEST\_SUPPORT**  
 280                   **ELSE NO\_OPTION**  
 281                   *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

282 **sched\_fifo8**  
 283           **IF PCTS\_sched\_setscheduler and PCTS\_sched\_yield THEN**  
 284               **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 285                   **TEST:** Under the SCHED\_FIFO policy, when a running process issues the *sched\_yield()*  
 286                               function, the process becomes the tail of the process list for its priority.  
 287                   **ELSE NO\_TEST\_SUPPORT**  
 288                   **ELSE NO\_OPTION**  
 289                   *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

290 **R\_1 TEST:** Under the SCHED\_FIFO policy, the position of a process within the process lists is affected only by  
 291 process scheduling events.  
 292 **NOTE:** There is no known portable test method for this assertion.  
 293 **SEE:** Assertions *sched\_fifo1*, *sched\_fifo2*, *sched\_fifo3*, *sched\_fifo4*, *sched\_fifo5*, *sched\_fifo6*,  
 294 *sched\_fifo7*, *sched\_fifo8* in §13.2.1.

295 **4**           **IF PCTS\_sched\_setscheduler and PCTS\_sched\_get\_priority\_max and PCTS\_sched\_get\_priority\_min**  
 296           **THEN**  
 297               **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 298                   **TEST:** Valid priorities under the SCHED\_FIFO policy, are within the range returned by  
 299                               the function *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* when  
 300                               SCHED\_FIFO is provided as the parameter.  
 301                   **TR:** Try the return values of *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()*. If  
 302                               *sched\_get\_priority\_max()* returns a value less than INT\_MAX, try  
 303                               *sched\_get\_priority\_max()+1*. If *sched\_get\_priority\_min()* returns a value greater than  
 304                               INT\_MIN, try *sched\_get\_priority\_min()-1*.  
 305                   **ELSE NO\_TEST\_SUPPORT**  
 306                   **ELSE NO\_OPTION**  
 307                   *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

308 **5**           **IF PCTS\_sched\_setscheduler THEN**  
 309               **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 310                   **TEST:** The SCHED\_FIFO policy has a priority range of at least 32 priorities.  
 311                   **ELSE NO\_TEST\_SUPPORT**  
 312                   **ELSE NO\_OPTION**  
 313                   *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

314 **13.2.2 SCHED\_RR**

315 **6 IF PCTS\_sched\_setscheduler and PCTS\_sched\_rr\_get\_interval THEN**

316 **IF PCTS\_GAP\_sched\_setscheduler THEN**

317 **TEST:** The SCHED\_RR policy is identical to the SCHED\_FIFO policy with the additional

318 condition that when the implementation detects that a running process has been

319 executing as a running process for a time period of the length returned by the

320 function *sched\_rr\_get\_interval()* or longer, the process becomes the tail of its

321 process list and the head of that process list is removed and made a running

322 process.

323 **ELSE NO\_TEST\_SUPPORT**

324 **ELSE NO\_OPTION**

325 *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

326 **7 IF PCTS\_sched\_setscheduler THEN**

327 **IF PCTS\_GAP\_sched\_setscheduler THEN**

328 **TEST:** A process under the SCHED\_RR policy that is preempted and subsequently

329 resumes execution as a running process completes the unexpired portion of its

330 round-robin-interval time period.

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

332 **ELSE NO\_TEST\_SUPPORT**

333 **ELSE NO\_OPTION**

334 *Conformance for sched\_policy: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

335 **8 IF PCTS\_sched\_setscheduler and PCTS\_sched\_get\_priority\_max and PCTS\_sched\_get\_priority\_min**

336 **THEN**

337 **IF PCTS\_GAP\_sched\_setscheduler THEN**

338 **TEST:** For the SCHED\_RR policy, valid priorities are within the range returned by the

339 functions *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* when

340 SCHED\_RR is provided as the parameter.

341 **TR:** Try the return values of *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()*. If

342 *sched\_get\_priority\_max()* returns a value less than INT\_MAX, try

343 *sched\_get\_priority\_max()+1*. If *sched\_get\_priority\_min()* returns a value greater than

344 INT\_MIN, try *sched\_get\_priority\_min()-1*.

345 **ELSE NO\_TEST\_SUPPORT**

346 **ELSE NO\_OPTION**

347 *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

348 **9 IF PCTS\_sched\_setscheduler THEN**

349 **IF PCTS\_GAP\_sched\_setscheduler THEN**

350 **TEST:** The SCHED\_RR has a priority range of at least 32 priorities.

351 **ELSE NO\_TEST\_SUPPORT**

352 **ELSE NO\_OPTION**

353 *Conformance for sched\_policy: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

354 **13.2.3 SCHED\_OTHER**

355 **D\_2 TEST:** A PCD.1b documents the behavior of the SCHED\_OTHER policy as described in the definition of

356 scheduling policy in §13.2.3.

357 *Conformance for sched\_policy: PASS*

358 **D\_3 TEST:** The PCD.1b documents the effect of scheduling processes with the SCHED\_OTHER policy as

359 described in a system in which other processes are executing under SCHED\_FIFO or SCHED\_RR, in

360 §13.2.3.

361 *Conformance for sched\_policy: PASS*

```

362 10      IF PCTS_sched_setscheduler and PCTS_sched_get_priority_max and PCTS_sched_get_priority_min
363      THEN
364          IF PCTS_GAP_sched_setscheduler THEN
365              TEST:      Priorities of processes executing under the SCHED_OTHER policy are restricted
366                        to the range returned by the functions sched_get_priority_max() and
367                        sched_get_priority_min() when SCHED_OTHER is provided as the parameter.
368              TR:      Try the return values of sched_get_priority_max() and sched_get_priority_min(). If
369                        sched_get_priority_max() returns a value less than INT_MAX, try
370                        sched_get_priority_max()+1. If sched_get_priority_min() returns a value greater than
371                        INT_MIN, try sched_get_priority_min()-1.
372          ELSE NO_TEST_SUPPORT
373      ELSE NO_OPTION
374      Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

```

## 375 13.3 Process Scheduling Functions

### 376 13.3.1 Set Scheduling Parameters

377 Function: *sched\_setparam()*

#### 378 13.3.1.1 Synopsis

```

379 1
380      M_GA_stdC_proto_decl(int; sched_setparam; pid_t pid, const struct sched_param *param;
381      sched.h;;;)
382      SEE:      Assertion GA_stdC_proto_decl in §2.7.3
383      Conformance for sched_setparam: PASS[1, 2], NO_OPTION

384 2
385      M_GA_commonC_int_result_decl(sched_setparam; sched.h;;;)
386      SEE:      Assertion GA_commonC_int_result_decl in §2.7.3
387      Conformance for sched_setparam: PASS[1, 2], NO_OPTION

388 3
389      M_GA_macro_result_decl(int; sched_setparam; sched.h;;;)
390      SEE:      Assertion GA_macro_result_decl in §1.3.4
391      Conformance for sched_setparam: PASS, NO_OPTION

392 4
393      M_GA_macro_args ( sched_setparam; sched.h;;;)
394      SEE:      Assertion GA_macro_args in §2.7.3
395      Conformance for sched_setparam: PASS, NO_OPTION

```

#### 396 13.3.1.2 Description

```

397 sched_setparam
398     IF PCTS_sched_setparam THEN
399         IF PCTS_GAP_sched_setparam THEN
400             TEST:      A successful call to sched_setparam() sets the scheduling parameters of the
401                       process specified by pid to the values specified by the sched_param structure
402                       pointed to by param, and returns the value 0.
403         ELSE NO_TEST_SUPPORT
404     ELSE NO_OPTION
405     Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

406 5      IF PCTS_sched_setparam THEN

```

```

407         IF PCTS_GAP_sched_setparam THEN
408             TEST: Any integer within the inclusive priority range for the current scheduling policy
409                 of the process specified by pid is a valid value of the sched_priority member in
410                 the param structure.
411             TR: Try the return values of sched_get_priority_max() and sched_get_priority_min(). If
412                 sched_get_priority_max() returns a value less than INT_MAX, try
413                 sched_get_priority_max()+1. If sched_get_priority_min() returns a value greater than
414                 INT_MIN, try sched_get_priority_min()-1.
415             ELSE NO_TEST_SUPPORT
416         ELSE NO_OPTION
417         Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

418     6     IF PCTS_sched_setparam THEN
419         IF PCTS_GAP_sched_setparam THEN
420             TEST: Higher numerical values for the priority represent higher priorities.
421             NOTE: There is no known portable test method for this assertion.
422         ELSE NO_TEST_SUPPORT
423     ELSE NO_OPTION
424     Conformance for sched_setparam: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

425     D_1 IF PCTS_sched_setparam a PCD.1b documents the following THEN
426         TEST: A PCD.1b that documents the behavior of sched_setparam() if the value of pid is
427                 negative, does so in §13.3.1.2.
428     ELSE NO_OPTION
429     Conformance for sched_setparam: PASS, NO_OPTION

430     7     IF PCTS_sched_setparam THEN
431         IF PCTS_GAP_sched_setparam THEN
432             TEST: When a process specified by pid exists, and if the calling process has
433                 permission, the scheduling parameters are set for the process whose process ID
434                 is equal to pid.
435         ELSE NO_TEST_SUPPORT
436     ELSE NO_OPTION
437     Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

438     8     IF PCTS_sched_setparam THEN
439         IF PCTS_GAP_sched_setparam THEN
440             TEST: When pid is zero, the scheduling parameters are set for the calling process.
441         ELSE NO_TEST_SUPPORT
442     ELSE NO_OPTION
443     Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

444     D_2 IF PCTS_sched_setparam THEN
445         TEST: A PCD.1b documents the conditions under which one process has permission to change
446                 the scheduling parameters of another process, in §13.3.1.2.
447     ELSE NO_OPTION
448     Conformance for sched_setparam: PASS, NO_OPTION

449     9     IF PCTS_sched_setparam THEN
450         IF PCTS_RAP_sched_setparam THEN
451             TEST: The requesting process must have the appropriate privilege to set its own
452                 scheduling parameters or those of another process.
453         ELSE NO_TEST_SUPPORT
454     ELSE NO_OPTION
455     Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

```

456 **D\_3 IF** *PCTS\_sched\_setparam* and a PCD.1b documents the following **THEN**  
 457       **TEST:**     A PCD.1b that documents whether the requesting process must have appropriate  
 458                     privilege to set its own scheduling parameters or those of another process, does so in  
 459                     §13.3.1.2.  
 460       **ELSE NO\_OPTION**  
 461       *Conformance for sched\_setparam: PASS, NO\_OPTION*

462 **10 IF** *PCTS\_sched\_setparam* **THEN**  
 463       **TEST:**     The target process, whether it is running or not running, resumes execution after all  
 464                     other runnable processes of equal or greater priority have been scheduled to run.  
 465       **NOTE:**     There is no known portable test method for this assertion.  
 466       **ELSE NO\_OPTION**  
 467       *Conformance for sched\_setparam: PASS, NO\_TEST, NO\_OPTION*

468 **11 IF** *PCTS\_sched\_setparam* **THEN**  
 469       **IF** *PCTS\_GAP\_sched\_setparam* **THEN**  
 470           **TEST:**     When the priority of the process specified by the *pid* argument is set higher  
 471                     than that of the lowest-priority running process and if the specified process is  
 472                     ready to run, the process specified by the *pid* argument preempts a lowest  
 473                     priority running process.  
 474           **NOTE:**     There is no known portable test method for this assertion.  
 475           **ELSE NO\_TEST\_SUPPORT**  
 476       **ELSE NO\_OPTION**  
 477       *Conformance for sched\_setparam: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

478 **12 IF** *PCTS\_sched\_setparam* **THEN**  
 479       **IF** *PCTS\_GAP\_sched\_setparam* **THEN**  
 480           **TEST:**     When the process calling *sched\_setparam()* sets its own priority lower than that  
 481                     of one or more other nonempty process lists, then the process that is the head of  
 482                     the highest priority list preempts the calling process.  
 483           **NOTE:**     There is no known portable test method for this assertion.  
 484           **ELSE NO\_TEST\_SUPPORT**  
 485       **ELSE NO\_OPTION**  
 486       *Conformance for sched\_setparam: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

487 **D\_4 IF** *PCTS\_sched\_setparam* **THEN**  
 488       **IF** *PCTS\_GAP\_sched\_setparam* **THEN**  
 489           **TEST:**     A PCD.1b documents the result if the current scheduling policy for the process  
 490                     specified by *pid* is not SCHED\_FIFO or SCHED\_RR, including SCHED\_OTHER, in  
 491                     §13.3.1.2.  
 492           **ELSE NO\_TEST\_SUPPORT**  
 493       **ELSE NO\_OPTION**  
 494       *Conformance for sched\_setparam: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

495 **D\_5 IF** *PCTS\_sched\_setparam* and a PCD.1b documents the following **THEN**  
 496       **TEST:**     A PCD.1b that documents whether or not it supports the *sched\_setparam()* function  
 497                     does so in §13.3.1.2.  
 498       **ELSE NO\_OPTION**  
 499       *Conformance for sched\_setparam: PASS, NO\_OPTION*

#### 500 13.3.1.4 Errors

501 **13 IF** *PCTS\_sched\_setparam* **THEN**  
 502       **IF** *PCTS\_GAP\_sched\_setparam* **THEN**  
 503           **TEST:**     A call to *sched\_setparam()*, when one or more of the requested scheduling  
 504                     parameters is outside the range defined for the scheduling policy of the specified  
 505                     *pid*, returns a value of -1 and sets *errno* to [EINVAL].

506                   **TR:** Try the return values of `sched_get_priority_max()` and `sched_get_priority_min()`. If  
507                   `sched_get_priority_max()` returns a value less than `INT_MAX`, try  
508                   `sched_get_priority_max()+1`. If `sched_get_priority_min()` returns a value greater than  
509                   `INT_MIN`, try `sched_get_priority_min()-1`.  
510                   **ELSE NO\_TEST\_SUPPORT**  
511                   **ELSE NO\_OPTION**  
512                   Conformance for `sched_setparam`: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

513   **14**           **IF** not `PCTS_sched_setparam` **THEN**  
514                   **TEST:**     A call to `sched_setparam()` returns a value of -1 and sets `errno` to `[ENOSYS]`.  
515                   **ELSE NO\_OPTION**  
516                   Conformance for `sched_setparam`: *PASS, NO\_OPTION*

517   **15**           **IF** `PCTS_sched_setparam` **THEN**  
518                   **IF** `PCTS_GAP_sched_setparam` **THEN**  
519                   **TEST:**     A call to `sched_setparam()` when the requesting process does not have  
520                   permission to set the scheduling parameters for the specified process, returns a  
521                   value of -1 and sets `errno` to `[EPERM]`.  
522                   **ELSE NO\_TEST\_SUPPORT**  
523                   **ELSE NO\_OPTION**  
524                   Conformance for `sched_setparam`: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

525   **16**           **IF** `PCTS_sched_setparam` **THEN**  
526                   **IF** `PCTS_RAP_sched_setparam` **THEN**  
527                   **TEST:**     A call to `sched_setparam()` when the requesting process does not have the  
528                   appropriate privilege to invoke `sched_setparam()`, returns a value of -1 and sets  
529                   `errno` to `[EPERM]`.  
530                   **ELSE NO\_TEST\_SUPPORT**  
531                   **ELSE NO\_OPTION**  
532                   Conformance for `sched_setparam`: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

533   **17**           **IF** `PCTS_sched_setparam` **THEN**  
534                   **IF** `PCTS_GAP_sched_setparam` **THEN**  
535                   **TEST:**     A call to `sched_setparam()`, when no process can be found corresponding to that  
536                   specified by `pid`, returns a value of -1 and sets `errno` to `[ESRCH]`.  
537                   **NOTE:**     A subroutine is recommended that returns a `pid` that doesn't correspond to any  
538                   existing process.  
539                   **ELSE NO\_TEST\_SUPPORT**  
540                   **ELSE NO\_OPTION**  
541                   Conformance for `sched_setparam`: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

### 542   **13.3.2**     **Get Scheduling Parameters**

543   Function: `sched_getparam()`

#### 544   **13.3.2.1**   **Synopsis**

545   **1**  
546        `M_GA_stdC_proto_decl(int; sched_getparam; pid_t pid, struct sched_param *param; sched.h;;)`  
547        **SEE:**     Assertion `GA_stdC_proto_decl` in §2.7.3  
548        Conformance for `sched_getparam`: *PASS[1, 2], NO\_OPTION*

549   **2**  
550        `M_GA_commonC_int_result_decl(sched_getparam; sched.h;;)`  
551        **SEE:**     Assertion `GA_commonC_int_result_decl` in §2.7.3  
552        Conformance for `sched_getparam`: *PASS[1, 2], NO\_OPTION*

553   **3**

554 *M\_GA\_macro\_result\_decl(int; sched\_getparam; sched.h;;)*  
555 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
556 *Conformance for sched\_getparam: PASS, NO\_OPTION*

557 **4**  
558 *M\_GA\_macro\_args ( sched\_getparam; sched.h;;)*  
559 **SEE:** Assertion GA\_macro\_args in §2.7.3  
560 *Conformance for sched\_getparam: PASS, NO\_OPTION*

561 **sched\_getparam**  
562 **IF** *PCTS\_sched\_getparam* **THEN**  
563 **TEST:** A successful call to *sched\_getparam()* returns the scheduling parameters of a process  
564 specified by *pid* in the *sched\_param* structure pointed to by *param*, and returns the  
565 value zero.  
566 **ELSE** *NO\_OPTION*  
567 *Conformance for sched\_getparam: PASS, NO\_OPTION*

568 **5** **IF** *PCTS\_sched\_getparam* **THEN**  
569 **TEST:** When a process specified by *pid* exists and if the calling process has permission, the  
570 scheduling parameters for the process whose process ID is equal to *pid* are returned.  
571 **ELSE** *NO\_OPTION*  
572 *Conformance for sched\_getparam: PASS, NO\_OPTION*

573 **6** **IF** *PCTS\_sched\_getparam* **THEN**  
574 **TEST:** When *pid* is zero, the scheduling parameters for the calling process are returned.  
575 **ELSE** *NO\_OPTION*  
576 *Conformance for sched\_getparam: PASS, NO\_OPTION*

577 **D\_1** **IF** *PCTS\_sched\_getparam* and PCD.1b documents the following **THEN**  
578 **TEST:** A PCD.1b that documents the behavior of *sched\_getparam()* if the value of *pid* is  
579 negative, does so in §13.3.2.2.  
580 **ELSE** *NO\_OPTION*  
581 *Conformance for sched\_getparam: PASS, NO\_OPTION*

582 **D\_2** **IF** *PCTS\_sched\_getparam* and PCD.1b documents the following **THEN**  
583 **TEST:** A PCD.1b that documents whether or not it supports the *sched\_getparam()* function  
584 does so in §13.3.2.2.  
585 **ELSE** *NO\_OPTION*  
586 *Conformance for sched\_getparam: PASS, NO\_OPTION*

587 **R\_1** **IF** *PCTS\_sched\_getparam* **THEN**  
588 **TEST:** When a call to *sched\_getparam()* completes successfully, the interface returns zero.  
589 **ELSE** *NO\_OPTION*  
590 **SEE:** Assertion *sched\_getparam* in §13.3.2.2

591 **R\_2** **IF** *PCTS\_sched\_getparam* **THEN**  
592 **TEST:** When a call to *sched\_getparam()* completes unsuccessfully, the interface returns a  
593 value of -1, and sets *errno* to indicate the error.  
594 **ELSE** *NO\_OPTION*  
595 **SEE:** All assertions in in §13.3.2.4

596 **13.3.2.4 Errors**

597 **7** **IF** not *PCTS\_sched\_getparam* **THEN**  
598 **TEST:** A call to *sched\_getparam()* returns a value of -1 and sets *errno* to [ENOSYS].  
599 **ELSE** *NO\_OPTION*  
600 *Conformance for sched\_getparam: PASS, NO\_OPTION*

601     **8**         **IF** *PCTS\_sched\_getparam* **THEN**  
602             **TEST:**     A call to *sched\_getparam*(), when the requesting process does not have permission to  
603                         obtain the scheduling parameters of the specified process, returns a value of -1 and  
604                         sets *errno* to [EPERM].  
605             **ELSE NO\_OPTION**  
606             Conformance for *sched\_getparam*: *PASS, NO\_OPTION*

607     **9**         **IF** *PCTS\_sched\_getparam* **THEN**  
608             **TEST:**     A call to *sched\_getparam*(), when No process can be found corresponding to that  
609                         specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].  
610             **NOTE:**     A subroutine is recommended that returns a *pid* that doesn't correspond to any  
611                         existing process.  
612             **ELSE NO\_OPTION**  
613             Conformance for *sched\_getparam*: *PASS, NO\_OPTION*

### 614     **13.3.3 Set Scheduling Policy and Scheduling Parameters**

615     Function: *sched\_setscheduler*()

616     **1**  
617         *M\_GA\_stdC\_proto\_decl(int; sched\_setscheduler; pid\_t pid, int policy, const struct sched\_param*  
618         *\*param; sched.h;);*  
619         **SEE:**         Assertion *GA\_stdC\_proto\_decl* in §2.7.3  
620         Conformance for *sched\_setscheduler*: *PASS[1, 2], NO\_OPTION*

621     **2**  
622         *M\_GA\_commonC\_int\_result\_decl(sched\_setscheduler; sched.h;);*  
623         **SEE:**         Assertion *GA\_commonC\_int\_result\_decl* in §2.7.3  
624         Conformance for *sched\_setscheduler*: *PASS[1, 2], NO\_OPTION*

625     **3**  
626         *M\_GA\_macro\_result\_decl(int; sched\_setscheduler; sched.h;);*  
627         **SEE:**         Assertion *GA\_macro\_result\_decl* in §1.3.4  
628         Conformance for *sched\_setscheduler*: *PASS, NO\_OPTION*

629     **4**  
630         *M\_GA\_macro\_args ( sched\_setscheduler; sched.h;);*  
631         **SEE:**         Assertion *GA\_macro\_args* in §2.7.3  
632         Conformance for *sched\_setscheduler*: *PASS, NO\_OPTION*

#### 633     **13.3.3.2 Description**

634     **sched\_setscheduler**  
635         **IF** *PCTS\_sched\_setscheduler* **THEN**  
636             **IF** *PCTS\_GAP\_sched\_setscheduler* **THEN**  
637                 **TEST:**     A successful call to *sched\_setscheduler*() sets the scheduling policy of the  
638                         process specified by *pid* to *policy*, and its scheduling parameters to and the  
639                         parameters specified in the *sched\_param* structure pointed to by *param*, and  
640                         returns the former scheduling policy of the specified process.  
641             **ELSE NO\_TEST\_SUPPORT**  
642             **ELSE NO\_OPTION**  
643             Conformance for *sched\_setscheduler*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

644     **5**         **IF** *PCTS\_sched\_setscheduler* **THEN**  
645             **IF** *PCTS\_GAP\_sched\_setscheduler* **THEN**  
646                 **TEST:**     The *sched\_priority* member in the *param* structure can take on any integer value  
647                         within the inclusive priority range for the scheduling policy specified by *policy*.



648                   **TR:** Try to return values of *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()*. If  
649                   *sched\_get\_priority\_max()* returns a value less than `INT_MAX`, try  
650                   *sched\_get\_priority\_max()+1*. If *sched\_get\_priority\_min()* returns a value greater than  
651                   `INT_MIN`, try *sched\_get\_priority\_min()-1*.  
652                   **ELSE NO\_TEST\_SUPPORT**  
653                   **ELSE NO\_OPTION**  
654                   Conformance for *sched\_setscheduler*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

655 **D\_1 IF** *PCTS\_sched\_setscheduler* and a PCD.1b documents the following **THEN**  
656                   **TEST:** A PCD.1b that documents the behavior of *sched\_setscheduler(pid, policy, param)*, if  
657                   the value of *pid* is negative, does so in §13.3.3.2.  
658                   **ELSE NO\_OPTION**  
659                   Conformance for *sched\_setscheduler*: *PASS, NO\_OPTION*

660 **6 IF** *PCTS\_sched\_setscheduler* **THEN**  
661                   **IF** *PCTS\_GAP\_sched\_setscheduler* **THEN**  
662                   **SETUP:** Include the header `<sched.h>`.  
663                   **TEST:** The possible values for the *policy* parameter, in the call *sched\_setscheduler*  
664                   (*pid, policy, param*), are defined.  
665                   **ELSE NO\_TEST\_SUPPORT**  
666                   **ELSE NO\_OPTION**  
667                   Conformance for *sched\_setscheduler*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

668 **7 IF** *PCTS\_sched\_setscheduler* **THEN**  
669                   **IF** *PCTS\_GAP\_sched\_setscheduler* **THEN**  
670                   **TEST:** When a process specified by *pid* exists, and if the calling process has  
671                   permission, *sched\_setscheduler (pid, policy, param)* sets the scheduling policy  
672                   for the process whose process ID is equal to *pid*.  
673                   **ELSE NO\_TEST\_SUPPORT**  
674                   **ELSE NO\_OPTION**  
675                   Conformance for *sched\_setscheduler*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

676 **8 IF** *PCTS\_sched\_setscheduler* **THEN**  
677                   **IF** *PCTS\_GAP\_sched\_setscheduler* **THEN**  
678                   **TEST:** When a process specified by *pid* exists, and if the calling process has  
679                   permission, *sched\_setscheduler (pid, policy, param)* sets the scheduling  
680                   parameters for the process whose process is equal to *pid*.  
681                   **ELSE NO\_TEST\_SUPPORT**  
682                   **ELSE NO\_OPTION**  
683                   Conformance for *sched\_setscheduler*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

684 **9 IF** *PCTS\_sched\_setscheduler* **THEN**  
685                   **IF** *PCTS\_GAP\_sched\_setscheduler* **THEN**  
686                   **TEST:** When *pid* is zero, *sched\_setscheduler (pid, policy, param)* sets the scheduling  
687                   policy for the calling process.  
688                   **ELSE NO\_TEST\_SUPPORT**  
689                   **ELSE NO\_OPTION**  
690                   Conformance for *sched\_setscheduler*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

691 **10 IF** *PCTS\_sched\_setscheduler* **THEN**  
692                   **IF** *PCTS\_GAP\_sched\_setscheduler* **THEN**  
693                   **TEST:** When *pid* is zero, *sched\_setscheduler(pid, policy, param)* sets the scheduling  
694                   parameters for the calling process.  
695                   **ELSE NO\_TEST\_SUPPORT**  
696                   **ELSE NO\_OPTION**  
697                   Conformance for *sched\_setscheduler*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

698 **D\_1 IF** *PCTS\_sched\_setscheduler* **THEN**

699                   **TEST:**     The PCD.1b documents the conditions under which one process has the appropriate  
 700   privilege to change the scheduling parameters of another process, in §13.3.3.2.  
 701                   **ELSE NO\_OPTION**  
 702                   *Conformance for sched\_setscheduler: PASS, NO\_OPTION*

703   **D\_2 IF PCTS\_sched\_setscheduler THEN**  
 704                   **TEST:**     The PCD.1b documents the conditions under which one process has the appropriate  
 705   privilege to change the scheduling parameters of another process, in §13.3.3.2.  
 706                   **ELSE NO\_TEST\_SUPPORT**  
 707                   **ELSE NO\_OPTION**  
 708                   *Conformance for sched\_setscheduler: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

709   **D\_3 IF PCTS\_sched\_setscheduler and a PCD.1b documents the following THEN**  
 710                   **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 711                                   **TEST:**     A PCD.1 that documents whether the requesting process has permission to set  
 712   its own scheduling parameters or those of another process, does so in §13.3.3.2.  
 713                                   **ELSE NO\_TEST\_SUPPORT**  
 714                   **ELSE NO\_OPTION**  
 715                   *Conformance for sched\_setscheduler: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

716   **D\_4 IF PCTS\_sched\_setscheduler and a PCD.1b documents the following THEN**  
 717                   **TEST:**     A PCD.1 that documents restrictions that apply as to the appropriate privileges  
 718   required to set a process's own scheduling policy, or another process's scheduling  
 719   policy, to a particular value, does so in §13.3.3.2.  
 720                   **ELSE NO\_OPTION**  
 721                   *Conformance for sched\_setscheduler: PASS, NO\_OPTION*

722   **R\_1 IF PCTS\_sched\_setscheduler THEN**  
 723                   **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 724                                   **TEST:**     The *sched\_setscheduler()* function is considered successful if it succeeds in  
 725   setting the scheduling policy and scheduling parameters of the process specified  
 726   by *pid* to the values specified by *policy* and the structure *param*, respectively.  
 727                                   **ELSE NO\_TEST\_SUPPORT**  
 728                   **ELSE NO\_OPTION**  
 729                   **SEE:**     Assertion *sched\_setscheduler* in §13.3.3.2

730   **D\_5 IF PCTS\_sched\_setscheduler and a PCD.1b documents the following THEN**  
 731                   **TEST:**     A PCD.1 that documents whether or not it supports the *sched\_setscheduler()* function,  
 732   does so in §13.3.3.2.  
 733                   **ELSE NO\_OPTION**  
 734                   *Conformance for sched\_setscheduler: PASS, NO\_OPTION*

### 735   **13.3.3.3 Returns**

736   **R\_2 IF PCTS\_sched\_setscheduler THEN**  
 737                   **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 738                                   **TEST:**     When a call to *sched\_setscheduler()* completes successfully, the interface  
 739   returns the former scheduling policy of the specified process.  
 740                                   **ELSE NO\_TEST\_SUPPORT**  
 741                   **ELSE NO\_OPTION**  
 742                   **SEE:**     Assertion *sched\_setscheduler* in §13.3.3.2

743   **R\_3 IF PCTS\_sched\_setscheduler THEN**  
 744                   **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 745                                   **TEST:**     When a call to *sched\_setscheduler()* completes unsuccessfully, the policy and  
 746   scheduling parameters remain unchanged, and the interface returns a value of  
 747   -1 and sets *errno* to indicate the error.

748                   **ELSE NO\_TEST\_SUPPORT**  
 749                   **ELSE NO\_OPTION**  
 750                   **SEE:**     All assertions in §13.3.3.4

### 751    **13.3.3.4 Errors**

752    **11**       **IF PCTS\_sched\_setscheduler THEN**  
 753               **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 754               **TEST:**    A call to *sched\_setscheduler()*, when the value of the *policy* parameter is  
 755                            invalid, returns a value of -1 and sets *errno* to [EINVAL].  
 756               **NOTE:**    A subroutine is recommended that either invalid value for *policy* or indicates  
 757                            that there is no way to generate invalid value for *policy* on the system.  
 758               **ELSE NO\_TEST\_SUPPORT**  
 759               **ELSE NO\_OPTION**  
 760               Conformance for *sched\_setscheduler*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

761    **12**       **IF PCTS\_sched\_setscheduler THEN**  
 762               **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 763               **TEST:**    A call to *sched\_setscheduler()*, when one or more of the parameters contained  
 764                            in *param* is outside the valid range for the specified scheduling policy, returns  
 765                            a value of -1 and sets *errno* to [EINVAL].  
 766               **TR:**      Try the return values of *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()*. If  
 767                            *sched\_get\_priority\_max()* returns a value less than INT\_MAX, try  
 768                            *sched\_get\_priority\_max()+1*. If *sched\_get\_priority\_min()* returns a value greater than  
 769                            INT\_MIN, try *sched\_get\_priority\_min()-1*.  
 770               **ELSE NO\_TEST\_SUPPORT**  
 771               **ELSE NO\_OPTION**  
 772               Conformance for *sched\_setscheduler*: *PASS, NO\_OPTION*

773    **13**       **IF not PCTS\_sched\_setscheduler THEN**  
 774               **TEST:**    A call to *sched\_setscheduler()*, returns a value of -1 and sets *errno* to [ENOSYS].  
 775               **ELSE NO\_OPTION**  
 776               Conformance for *sched\_setscheduler*: *PASS, NO\_OPTION*

777    **14**       **IF PCTS\_sched\_setscheduler THEN**  
 778               **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 779               **TEST:**    A call to *sched\_setscheduler()*, when the requesting process does not have  
 780                            permission to set the scheduling parameters of the specified process, returns a  
 781                            value of -1 and sets *errno* to [EPERM].  
 782               **ELSE NO\_TEST\_SUPPORT**  
 783               **ELSE NO\_OPTION**  
 784               Conformance for *sched\_setscheduler*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

785    **15**       **IF PCTS\_sched\_setscheduler THEN**  
 786               **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 787               **TEST:**    A call to *sched\_setscheduler()*, when the requesting process does not have  
 788                            permission to set the scheduling policy of the specified process, returns a value  
 789                            of -1 and sets *errno* to [EPERM].  
 790               **ELSE NO\_TEST\_SUPPORT**  
 791               **ELSE NO\_OPTION**  
 792               Conformance for *sched\_setscheduler*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

793    **16**       **IF PCTS\_sched\_setscheduler THEN**  
 794               **IF PCTS\_GAP\_sched\_setscheduler THEN**  
 795               **TEST:**    A call to *sched\_setscheduler()*, when no process can be found corresponding to  
 796                            that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].  
 797               **ELSE NO\_TEST\_SUPPORT**

798           **ELSE NO\_OPTION**  
799           *Conformance for sched\_getscheduler: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

800   **13.3.4    Get Scheduling Policy**

801   Function: *sched\_getscheduler()*

802   **13.3.4.1 Synopsis**

803   **1**  
804       *M\_GA\_stdC\_proto\_decl(int; sched\_getscheduler; pid\_t pid; sched.h;;)*  
805       **SEE:**     Assertion GA\_stdC\_proto\_decl in §2.7.3  
806       *Conformance for sched\_getscheduler: PASS[1, 2], NO\_OPTION*

807   **2**  
808       *M\_GA\_commonC\_int\_result\_decl(sched\_getscheduler; sched.h;;)*  
809       **SEE:**     Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
810       *Conformance for sched\_getscheduler: PASS[1, 2], NO\_OPTION*

811   **3**  
812       *M\_GA\_macro\_result\_decl(int; sched\_getscheduler; sched.h;;)*  
813       **SEE:**     Assertion GA\_macro\_result\_decl in §1.3.4  
814       *Conformance for sched\_getscheduler: PASS, NO\_OPTION*

815   **4**  
816       *M\_GA\_macro\_args ( sched\_getscheduler; sched.h;;)*  
817       **SEE:**     Assertion GA\_macro\_args in §2.7.3  
818       *Conformance for sched\_getscheduler: PASS, NO\_OPTION*

819   **13.3.4.2 Description**

820   **sched\_getscheduler**

821       **IF PCTS\_sched\_getscheduler THEN**  
822           **TEST:**    A successful call to *sched\_getscheduler()* returns the scheduling policy of the process  
823                        specified by *pid*.  
824       **ELSE NO\_OPTION**  
825       *Conformance for sched\_getscheduler: PASS, NO\_OPTION*

826   **5**       **IF PCTS\_sched\_getscheduler and a PCD.1b documents the following THEN**  
827           **TEST:**    A PCD.1b that documents the behavior of *sched\_getscheduler()* if the value of *pid* is  
828                        negative, does so in §13.3.4.2.  
829       **ELSE NO\_OPTION**  
830       *Conformance for sched\_getscheduler: PASS, NO\_OPTION*

831   **6**       **IF PCTS\_sched\_getscheduler THEN**  
832           **TEST:**    The values that can be returned by *sched\_getscheduler()* are defined in the header file  
833                        <sched.h> .  
834       **ELSE NO\_OPTION**  
835       *Conformance for sched\_getscheduler: PASS, NO\_OPTION*

836   **7**       **IF PCTS\_sched\_getscheduler THEN**  
837           **TEST:**    When a process specified by *pid* exists, and if the calling process has permission, the  
838                        scheduling policy is returned for the process whose process ID is equal to *pid*.  
839       **ELSE NO\_OPTION**  
840       *Conformance for sched\_getscheduler: PASS, NO\_OPTION*

841 **8** **IF** *PCTS\_sched\_getscheduler* **THEN**  
842 **TEST:** When *pid* is zero, the scheduling policy is returned for the calling process.  
843 **ELSE NO\_OPTION**  
844 *Conformance for sched\_getscheduler: PASS, NO\_OPTION*

845 **D\_1** **IF** *PCTS\_sched\_getscheduler* and a PCD.1b documents the following **THEN**  
846 **TEST:** A PCD.1b that documents whether or not it supports the *sched\_getscheduler()* function  
847 does so in §13.3.4.2.  
848 **ELSE NO\_OPTION**  
849 *Conformance for sched\_getscheduler: PASS, NO\_OPTION*

850 **13.3.4.3 Returns**

851 **R\_1** **IF** *PCTS\_sched\_getscheduler* **THEN**  
852 **TEST:** When a call to *sched\_getscheduler()* completes successfully, the interface returns the  
853 former policy of the specified process.  
854 **ELSE NO\_OPTION**  
855 **SEE:** Assertion *sched\_getscheduler* in §13.3.4.2

856 **R\_2** **IF** *PCTS\_sched\_getscheduler* **THEN**  
857 **TEST:** When a call to *sched\_getscheduler()* completes unsuccessfully, the interface returns  
858 a value of -1, and sets *errno* to indicate the error.  
859 **ELSE NO\_OPTION**  
860 **SEE:** All assertions in §13.3.4.4

861 **13.3.4.4 Errors**

862 **9** **IF** not *PCTS\_sched\_getscheduler* **THEN**  
863 **TEST:** A call to *sched\_getscheduler()* returns a value of -1 and sets *errno* to [ENOSYS].  
864 **ELSE NO\_OPTION**  
865 *Conformance for sched\_getscheduler: PASS, NO\_OPTION*

866 **10** **IF** *PCTS\_sched\_getscheduler* **THEN**  
867 **TEST:** A call to *sched\_getscheduler()* when the requesting process does not have permission  
868 to determine the scheduling policy of the specified process, returns a value of -1 and  
869 sets *errno* to [EPERM].  
870 **ELSE NO\_OPTION**  
871 *Conformance for sched\_getscheduler: PASS, NO\_OPTION*

872 **11** **IF** *PCTS\_sched\_getscheduler* **THEN**  
873 **TEST:** A call to *sched\_getscheduler()* when no process can be found corresponding to that  
874 specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].  
875 **NOTE:** A subroutine is recommended that returns a *pid* that doesn't correspond to any  
876 existing process.  
877 **ELSE NO\_OPTION**  
878 *Conformance for sched\_getscheduler: PASS, NO\_OPTION*

879 **13.3.5 Yield Processor**

880 Function: *sched\_yield()*

881 **13.3.5.1 Synopsis**

882 **1**  
883 *M\_GA\_stdC\_proto\_decl(int; sched\_yield; sched.h;;)*  
884 **SEE:** Assertion *GA\_stdC\_proto\_decl* in §2.7.3  
885 *Conformance for sched\_yield: PASS[1, 2], NO\_OPTION*

886 **2**  
 887 *M\_GA\_commonC\_int\_result\_decl(sched\_yield; sched.h;;)*  
 888 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 889 *Conformance for sched\_yield: PASS[1, 2], NO\_OPTION*

890 **3**  
 891 *M\_GA\_macro\_result\_decl(int; sched\_yield; sched.h;;)*  
 892 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 893 *Conformance for sched\_yield: PASS, NO\_OPTION*

894 **4**  
 895 *M\_GA\_macro\_args ( sched\_yield; sched.h;;)*  
 896 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 897 *Conformance for sched\_yield: PASS, NO\_OPTION*

### 898 13.3.5.2 Description

899 **sched\_yield**  
 900 **IF** *PCTS\_sched\_yield* **THEN**  
 901 **TEST:** A successful call to *9sched\_yield()* forces the running process to relinquish the  
 902 process until it again becomes the head of its process list, and returns the value zero.  
 903 **NOTE:** There is no known portable test method for this assertion.  
 904 **ELSE** *NO\_OPTION*  
 905 *Conformance for sched\_yield: PASS, NO\_TEST, NO\_OPTION*

906 **D\_1** **IF** *PCTS\_sched\_yield* and a PCD.1b documents the following **THEN**  
 907 **TEST:** A PCD.1b that documents whether or not it supports the *sched\_yield()* function does  
 908 so in §13.3.5.2.  
 909 **ELSE** *NO\_OPTION*  
 910 *Conformance for sched\_yield: PASS, NO\_OPTION*

### 911 13.3.5.3 Returns

912 **R\_1** **IF** *PCTS\_sched\_yield* **THEN**  
 913 **TEST:** When a call to *sched\_yield()* completes successfully, the interface returns 0.  
 914 **ELSE** *NO\_OPTION*  
 915 **SEE:** Assertion *sched\_yield* in §13.3.5.2

916 **R\_2** **IF** *PCTS\_sched\_yield* **THEN**  
 917 **TEST:** When a call to *sched\_yield()* completes unsuccessfully, the interface returns a value  
 918 of -1, and sets *errno* to indicate the error.  
 919 **ELSE** *NO\_OPTION*  
 920 **SEE:** All assertions in §13.3.5.4

### 921 13.3.5.4 Errors

922 **5** **IF** not *PCTS\_sched\_yield* **THEN**  
 923 **TEST:** A call to *sched\_yield()* returns a value of -1 and sets *errno* to [ENOSYS]  
 924 **ELSE** *NO\_OPTION*  
 925 *Conformance for sched\_yield: PASS, NO\_OPTION*

### 926 13.3.6 Get Scheduling Parameter Limits

927 Functions:  
 928 *sched\_get\_priority\_max()*,

929 *sched\_get\_priority\_min()*,  
 930 *sched\_rr\_get\_interval()*

### 931 13.3.6.1 Synopsis

932 **1**

933 *M\_GA\_stdC\_proto\_decl(int; sched\_get\_priority\_max; int policy; sched.h;;)*  
 934 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 935 *Conformance for sched\_get\_priority\_max: PASS[1, 2], NO\_OPTION*

936 **2**

937 *M\_GA\_commonC\_int\_result\_decl(sched\_get\_priority\_max; sched.h;;)*  
 938 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 939 *Conformance for sched\_get\_priority\_max: PASS[1, 2], NO\_OPTION*

940 **3**

941 *M\_GA\_macro\_result\_decl(int; sched\_get\_priority\_max; sched.h;;)*  
 942 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 943 *Conformance for sched\_get\_priority\_max: PASS, NO\_OPTION*

944 **4**

945 *M\_GA\_macro\_args ( sched\_get\_priority\_max; sched.h;;)*  
 946 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 947 *Conformance for sched\_get\_priority\_max: PASS, NO\_OPTION*

948 **5**

949 *M\_GA\_stdC\_proto\_decl(int; sched\_get\_priority\_min; , int policy; sched.h;;)*  
 950 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 951 *Conformance for sched\_get\_priority\_min: PASS[5,6], NO\_OPTION*

952 **6**

953 *M\_GA\_commonC\_int\_result\_decl(sched\_get\_priority\_min; , int policy; sched.h;;)*  
 954 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 955 *Conformance for sched\_get\_priority\_min: PASS[5, 6], NO\_OPTION*

956 **7**

957 *M\_GA\_macro\_result\_decl(int; sched\_get\_priority\_min; sched.h;;)*  
 958 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 959 *Conformance for sched\_get\_priority\_min: PASS, NO\_OPTION*

960 **8**

961 *M\_GA\_macro\_args ( sched\_get\_priority\_min; sched.h;;)*  
 962 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 963 *Conformance for sched\_get\_priority\_min: PASS, NO\_OPTION*

964 **9**

965 *M\_GA\_stdC\_proto\_decl(int; sched\_rr\_get\_interval; , pid\_t pid, struct timespec \*interval;*  
 966 *sched.h;;)*  
 967 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 968 *Conformance for sched\_rr\_get\_interval: PASS[9, 10], NO\_OPTION*

969 **10**

970 *M\_GA\_commonC\_int\_result\_decl(sched\_rr\_get\_interval; , pid\_t pid, struct timespec \*interval;*  
 971 *sched.h;;)*  
 972 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 973 *Conformance for sched\_rr\_get\_interval: PASS[9, 10], NO\_OPTION*

974 **11**

975 *M\_GA\_macro\_result\_decl(int; sched\_rr\_get\_interval; sched.h;;)*

976           **SEE:**       Assertion GA\_macro\_result\_decl in §1.3.4  
977           *Conformance for sched\_rr\_get\_interval: PASS, NO\_OPTION*

978    **12**

979           *M\_GA\_macro\_args ( sched\_rr\_get\_interval; sched.h;;;)*  
980           **SEE:**       Assertion GA\_macro\_args in §2.7.3  
981           *Conformance for sched\_rr\_get\_interval: PASS, NO\_OPTION*

982    **13.3.6.2 Description**

983    **sched\_get\_priority\_max**  
984           **IF** PCTS\_sched\_get\_priority\_max **THEN**  
985            **TEST:**     A successful call to *sched\_get\_priority\_max()* returns the appropriate maximum for  
986                        the scheduling policy specified by *policy*.  
987            **TR:**     Test for SCHED\_FIFO, SCHED\_RR, and SCHED\_OTHER  
988            **ELSE NO\_OPTION**  
989            *Conformance for sched\_get\_priority\_max: PASS, NO\_OPTION*

990    **sched\_get\_priority\_min**  
991           **IF** PCTS\_sched\_get\_priority\_min **THEN**  
992            **TEST:**     A successful call to *sched\_get\_priority\_min ()* returns the appropriate minimum for  
993                        the scheduling policy specified by *policy*.  
994            **TR:**     Test for SCHED\_FIFO, SCHED\_RR, and SCHED\_OTHER  
995            **ELSE NO\_OPTION**  
996            *Conformance for sched\_get\_priority\_min: PASS, NO\_OPTION*

997    **sched\_rr\_get\_interval**  
998           **IF** PCTS\_sched\_rr\_get\_interval **THEN**  
999            **TEST:**     A successful call to *sched\_rr\_get\_interval()* updates the *timespec* structure referenced  
1000                        by the *interval* argument to contain the current execution time limit (i.e., time  
1001                        quantum) for the process specified by *pid*, and returns the value zero.  
1002            **NOTE:**     There is no known portable test method for this assertion.  
1003            **ELSE NO\_OPTION**  
1004            *Conformance for sched\_rr\_get\_interval: PASS, NO\_TEST, NO\_OPTION*

1005    **13**    **IF** PCTS\_sched\_rr\_get\_interval **THEN**  
1006            **TEST:**     When *pid* is zero, *sched\_rr\_get\_interval()* returns the current execution time limit for  
1007                        the calling process.  
1008            **NOTE:**     There is no known portable test method for this assertion.  
1009            **ELSE NO\_OPTION**  
1010            *Conformance for sched\_rr\_get\_interval: PASS, NO\_TEST, NO\_OPTION*

1011    **14**    **IF** PCTS\_sched\_get\_priority\_max **THEN**  
1012            **TEST:**     In the call to *sched\_get\_priority\_max(policy)*, the value of *policy* is one of the  
1013                        scheduling policy values defined in *<sched.h>*.  
1014            **NOTE:**     A subroutine is recommended that either returns an invalid scheduling policy or  
1015                        indicates that there is no way to generate an invalid scheduling policy on the system.  
1016            **ELSE NO\_OPTION**  
1017            *Conformance for sched\_get\_priority\_max: PASS, NO\_OPTION*

1018    **15**    **IF** PCTS\_sched\_get\_priority\_min **THEN**  
1019            **TEST:**     In the call *sched\_get\_priority\_min(policy)*, the value of *policy* is one of the scheduling  
1020                        policy values defined in *<sched.h>*.  
1021            **NOTE:**     A subroutine is recommended that either returns an invalid scheduling policy or  
1022                        indicates that there is no way to generate an invalid scheduling policy on the system.  
1023            **ELSE NO\_OPTION**  
1024            *Conformance for sched\_get\_priority\_min: PASS, NO\_OPTION*



1025 **D\_1 IF** *PCTS\_sched\_get\_priority\_max* and a PCD.1b documents the following **THEN**  
 1026       **TEST:** A PCD.1b that documents whether or not it supports the *sched\_get\_priority\_max()*  
 1027                   function does so in §13.3.6.2.  
 1028       **ELSE NO\_OPTION**  
 1029       *Conformance for sched\_get\_priority\_max: PASS, NO\_OPTION*

1030 **D\_2 IF** *PCTS\_sched\_get\_priority\_min* and a PCD.1b documents the following **THEN**  
 1031       **TEST:** A PCD.1b that documents whether or not it supports the *sched\_get\_priority\_min()*  
 1032                   function does so in §13.3.6.2.  
 1033       **ELSE NO\_OPTION**  
 1034       *Conformance for sched\_get\_priority\_min: PASS, NO\_OPTION*

1035 **D\_3 IF** *PCTS\_sched\_rr\_get\_interval* and a PCD.1b documents the following **THEN**  
 1036       **TEST:** A PCD.1b that documents whether or not it supports the *sched\_rr\_get\_interval()*  
 1037                   function does so in §13.3.6.2.  
 1038       **ELSE NO\_OPTION**  
 1039       *Conformance for sched\_rr\_get\_interval: PASS, NO\_OPTION*

### 1040 13.3.6.3 Returns

1041 **R\_1 IF** *PCTS\_sched\_get\_priority\_max* **THEN**  
 1042       **TEST:** When a call to *sched\_get\_priority\_max()* completes successfully, the interface returns  
 1043                   the appropriate maximum value.  
 1044       **ELSE NO\_OPTION**  
 1045       **SEE:** Assertion *sched\_get\_priority\_max* in §13.3.6.2

1046 **R\_2 IF** *PCTS\_sched\_get\_priority\_min* **THEN**  
 1047       **TEST:** When a call to *sched\_get\_priority\_min()* completes successfully, the interface returns  
 1048                   the appropriate minimum value.  
 1049       **ELSE NO\_OPTION**  
 1050       **SEE:** Assertion *sched\_get\_priority\_min* in §13.3.6.2

1051 **R\_3 IF** *PCTS\_sched\_get\_priority\_max* **THEN**  
 1052       **TEST:** When a call to *sched\_get\_priority\_max()* completes successfully, the interface returns  
 1053                   a value of -1, and sets *errno* to indicate the error.  
 1054       **ELSE NO\_OPTION**  
 1055       **SEE:** All assertions in §13.3.6.4

1056 **R\_4 IF** *PCTS\_sched\_get\_priority\_min* **THEN**  
 1057       **TEST:** When a call to *sched\_get\_priority\_min()* completes unsuccessfully, the interface  
 1058                   returns a value of -1, and sets *errno* to indicate the error.  
 1059       **ELSE NO\_OPTION**  
 1060       **SEE:** All assertions in §13.3.6.4

1061 **R\_5 IF** *PCTS\_sched\_rr\_get\_interval* **THEN**  
 1062       **TEST:** When a call to *sched\_rr\_get\_interval()*, completes successfully, the interface returns  
 1063                   zero.  
 1064       **ELSE NO\_OPTION**  
 1065       **SEE:** Assertion *sched\_rr\_get\_interval* in §13.3.6.2

1066 **R\_6 IF** *PCTS\_sched\_rr\_get\_interval* **THEN**  
 1067       **TEST:** When a call to *sched\_rr\_get\_interval()* completes unsuccessfully, the interface returns  
 1068                   a value of -1, and sets *errno* to indicate the error.  
 1069       **ELSE NO\_OPTION**  
 1070       **SEE:** All assertions in §13.3.6.4

### 1071 13.3.6.4 Errors

- 1072 **14** **IF** *PCTS\_sched\_get\_priority\_max* **THEN**  
 1073 **TEST:** A call to *sched\_get\_priority\_max()* when the value of the *policy* parameter does not  
 1074 represent a defined scheduling policy, returns a value of -1 and sets *errno* to [EINVAL].  
 1075 **NOTE:** A subroutine is recommended that either returns an invalid scheduling policy or  
 1076 indicates that there is no way to generate an invalid scheduling policy on the system.  
 1077 **ELSE NO\_OPTION**  
 1078 *Conformance for sched\_get\_priority\_max: PASS, NO\_OPTION*
- 1079 **17** **IF** *PCTS\_sched\_get\_priority\_min* **THEN**  
 1080 **TEST:** A call to *sched\_get\_priority\_min()* when the value of the *policy* parameter does not  
 1081 represent a defined scheduling policy, returns a value of -1 and sets *errno* to [EINVAL].  
 1082 **NOTE:** A subroutine is recommended that either returns an invalid scheduling policy or  
 1083 indicates that there is no way to generate an invalid scheduling policy on the system.  
 1084 **ELSE NO\_OPTION**  
 1085 *Conformance for sched\_get\_priority\_min: PASS, NO\_OPTION*
- 1086 **18** **IF** not *PCTS\_sched\_get\_priority\_max* **THEN**  
 1087 **TEST:** A call to *sched\_get\_priority\_max()* returns a value of -1 and sets *errno* to [ENOSYS]  
 1088 **ELSE NO\_OPTION**  
 1089 *Conformance for sched\_get\_priority\_max: PASS, NO\_OPTION*
- 1090 **19** **IF** not *PCTS\_sched\_get\_priority\_min* **THEN**  
 1091 **TEST:** A call to *sched\_get\_priority\_min()* returns a value of -1 and sets *errno* to [ENOSYS].  
 1092 **ELSE NO\_OPTION**  
 1093 *Conformance for sched\_get\_priority\_min: PASS, NO\_OPTION*
- 1094 **20** **IF** not *PCTS\_sched\_rr\_get\_interval* **THEN**  
 1095 **TEST:** A call to *sched\_rr\_get\_interval()* returns a value of -1 and sets *errno* to [ENOSYS].  
 1096 **ELSE NO\_OPTION**  
 1097 *Conformance for sched\_rr\_get\_interval: PASS, NO\_OPTION*
- 1098 **21** **IF** *PCTS\_sched\_get\_priority\_max* **THEN**  
 1099 **TEST:** A call to *sched\_get\_priority\_max()*, when no process can be found corresponding to  
 1100 that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH]  
 1101 **ELSE NO\_OPTION**  
 1102 *Conformance for sched\_get\_priority\_max: PASS, NO\_OPTION*
- 1103 **22** **IF** *PCTS\_sched\_get\_priority\_min* **THEN**  
 1104 **TEST:** A call to *sched\_get\_priority\_min()*, when no process can be found corresponding to  
 1105 that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].  
 1106 **NOTE:** A subroutine is recommended that returns a *pid* that doesn't correspond to any  
 1107 existing process.  
 1108 **ELSE NO\_OPTION**  
 1109 *Conformance for sched\_get\_priority\_min: PASS, NO\_OPTION*
- 1110 **23** **IF** *PCTS\_sched\_rr\_get\_interval* **THEN**  
 1111 **TEST:** A call to *sched\_rr\_get\_interval()*, when no process can be found corresponding to that  
 1112 specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH]  
 1113 **NOTE:** A subroutine is recommended that returns a *pid* that doesn't correspond to any  
 1114 existing process.  
 1115 **ELSE NO\_OPTION**  
 1116 *Conformance for sched\_rr\_get\_interval: PASS, NO\_OPTION*

## Section 14: Clocks and Timers

### 180 14.1 Data Definitions for Clocks and Timers

#### 181 14.1.1 Time Value Specification Structures

182 1 **SETUP:** Include the header `<time.h>`.  
 183 **TEST:** The structure *timespec* is defined, and includes the members

Member Type	Member Name	Description
<i>time_t</i>	<i>tv_sec</i>	Seconds
<i>long</i>	<i>tv_nsec</i>	Nanoseconds

188 *Conformance for timer\_hdr: PASS*

189 **D\_1 IF** a PCD.1b documents the following **THEN**  
 190 **TEST:** A PCD.1b that documents extensions to *timespec*, as permitted in POSIX.1b{3} §1.3.1.1  
 191 item (2), does so in §14.1.1.  
 192 **ELSE NO\_OPTION**  
 193 *Conformance for timer\_hdr: PASS, NO\_OPTION*

194 2 **SETUP:** Include the header `<time.h>`.  
 195 **TEST:** Extensions to *timespec* that may change the behavior of the application with respect to this  
 196 standard when those fields in the structure are uninitialized, are enabled as required by  
 197 POSIX.1b {3} §1.3.1.1.  
 198 **NOTE:** The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write  
 199 a portable test.  
 200 *Conformance for timer\_hdr: PASS, NO\_TEST*

201 3 **TEST:** The *tv\_nsec* member of *timespec* is only valid if less than the number of nanoseconds in a  
 202 second (1000 million).  
 203 *Conformance for timer\_hdr: PASS*

204 4 **IF**  $\{int\_max\} \leq 10E9$  **THEN**  
 205 **TEST:** The *tv\_nsec* member of *timespec* is only valid if less than the number of nanoseconds  
 206 in a second (1000 million).  
 207 **ELSE NO\_TEST\_SUPPORT**  
 208 *Conformance for timer\_hdr: PASS, NO\_TEST\_SUPPORT*

209 5 **TEST:** The time interval described by *timespec* is  $(tv\_sec \times 10^9 + tv\_nsec)$  nanoseconds.  
 210 *Conformance for timer\_hdr: PASS*

211 6 **SETUP:** Include the header `<time.h>`.  
 212 **TEST:** The structure *itimerspec* is defined, and includes the members

	<b>Member Type</b>	<b>Member Name</b>	<b>Description</b>
213			
214			
215	<i>struct timespec</i>	<i>it_interval</i>	Timer period
216	<i>struct timespec</i>	<i>it_value</i>	Timer expiration
217	<i>Conformance for timer_hdr: PASS</i>		
218	<b>D_2 IF</b> a PCD.1b documents the following <b>THEN</b>		
219	<b>TEST:</b>	A PCD.1b documents extensions to <i>itimerspec</i> , as permitted in POSIX1b{3} §1.3.1.1	
220		item (2), does so in §14.1	
221	<i>Conformance for timer_hdr: PASS, NO_OPTION</i>		
222	<b>7</b>	<b>SETUP:</b>	Include the header <code>&lt;time.h&gt;</code> .
223		<b>TEST:</b>	Extensions to <i>itimerspec</i> that may change the behavior of the application with respect to this
224			standard when those fields in the structure are uninitialized, are enabled as required by
225			POSIX.1b {3} §1.3.1.1.
226		<b>NOTE:</b>	The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write
227			a portable test.
228	<i>Conformance for timer_hdr: PASS, NO_TEST</i>		
229	<b>8</b>	<b>TEST:</b>	When the value described by <i>it_value</i> is nonzero, it indicates the time to or time of the next
230			timer expiration (for relative and absolute timer values, respectively).
231	<i>Conformance for timer_hdr: PASS</i>		
232	<b>9</b>	<b>TEST:</b>	When the value described by <i>it_value</i> is zero, the timer is disarmed..
233	<i>Conformance for timer_hdr: PASS</i>		
234	<b>10</b>	<b>TEST:</b>	When the value described by <i>it_interval</i> is nonzero, it specifies an interval to be used in
235			reloading the timer when it expires – that is, a periodic timer is specified.
236	<i>Conformance for timer_hdr: PASS</i>		
237	<b>11</b>	<b>TEST:</b>	When the value described by <i>it_interval</i> is zero, the timer is disarmed after its next
238			expiration – that is, a “one-shot” timer is specified.
239	<i>Conformance for timer_hdr: PASS</i>		
240	<b>14.1.2</b>	<b>Timer Event Notification Control Block</b>	
241	<b>12</b>	<b>IF</b> <code>{_POSIX_REALTIME_SIGNALS}</code> <b>THEN</b>	
242		<b>TEST:</b>	Per-process timers can be created that notify the process of timer expirations by
243			queuing a realtime extended signal.
244	<b>ELSE NO_TEST_SUPPORT</b>		
245	<i>Conformance for timer_hdr: PASS, NO_TEST_SUPPORT</i>		
246			
247	<b>14.1.3</b>	<b>Type Definitions</b>	
248	<b>13</b>	<b>SETUP:</b>	Include the header <code>&lt;sys/types.h&gt;</code> .
249		<b>TEST:</b>	The types <i>clockid_t</i> and <i>timer_t</i> are defined.
250	<i>Conformance for timer_hdr: PASS</i>		
251	<b>14.1.4</b>	<b>Manifest Constants</b>	
252	<b>14</b>	<b>SETUP:</b>	Include the header <code>&lt;time.h&gt;</code> .
253		<b>TEST:</b>	The constants <code>CLOCK_REALTIME</code> and <code>TIMER_ABSTIME</code> are defined.
254	<i>Conformance for timer_hdr: PASS</i>		

- 255 **15**     **SETUP:** Include the header `<unistd.h>`.  
 256     **TEST:**    The constant `{_POSIX_CLOCKRES_MIN}` and is defined as 20 ms (1/50 of a second).  
 257     *Conformance for timer\_hdr: PASS*
- 258 **16**     **TEST:**    The maximum allowable resolution for the `CLOCK_REALTIME` clock and all times based on  
 259                this clock, including the `nanosleep()` function, is `{_POSIX_CLOCKRES_MIN}`.  
 260     *Conformance for timer\_hdr: PASS*
- 261 **D\_3** **IF** a PCD.1b documents the following **THEN**  
 262         **TEST:**    A PCD.1b that documents support for smaller values of resolution for the  
 263                    `CLOCK_REALTIME` clock to provide finer granularity time bases, does so in §14.1.4.  
 264         **ELSE NO\_OPTION**  
 265         *Conformance for timer\_hdr: PASS, NO\_OPTION*
- 266 **17**     **TEST:**    The actual resolution for a specific clock is obtained using functions defined in this chapter.  
 267     *Conformance for timer\_hdr: PASS*
- 268 **D\_4** **IF** a PCD.1b documents the following **THEN**  
 269         **TEST:**    A PCD.1b that documents does so in §14.1.4.  
 270         **ELSE NO\_OPTION**  
 271         *Conformance for timer\_hdr: PASS, NO\_OPTION*
- 272 **D\_5** **TEST:**    The PCD.1b documents the actual resolution supported for the `nanosleep()` function of timers based  
 273                on this clock, if it differs from the resolution supported for the clock, in §14.1.4.  
 274     *Conformance for timer\_hdr: PASS*
- 275 **18**     **TEST:**    The maximum value of `CLOCK_REALTIME` clock and absolute timers based on it is at least that  
 276                defined by the C Standard [2] for the `time_t` type.  
 277     *Conformance for timer\_hdr: PASS*
- 278 **D\_6** **TEST:**    The PCD.1b documents the difference between the maximum value supported by the `nanosleep()`  
 279                function or timers based on this clock and the maximum value supported by the clock, if such a  
 280                difference exists, in §14.1.4.  
 281     *Conformance for timer\_hdr: PASS*

## 282 14.2 Clocks and Timer Functions

### 283 14.2.1 Clocks

284 Functions: `clock_settime()`, `clock_gettime()`, `clock_getres()`

#### 285 14.2.1.1 Synopsis

- 286 **1**  
 287     *M\_GA\_stdC\_proto\_decl(int; clock\_settime; clockid\_t clock\_id, const struct timespec \*tp; time.h;;)*  
 288     **SEE:**        Assertion `GA_stdC_proto_decl` in §2.7.3  
 289     *Conformance for clock\_settime: PASS[1, 2], NO\_OPTION*
- 290 **2**  
 291     *M\_GA\_commonC\_int\_result\_decl(clock\_settime; time.h;;)*  
 292     **SEE:**        Assertion `GA_commonC_int_result_decl` in §2.7.3  
 293     *Conformance for clock\_settime: PASS[1, 2], NO\_OPTION*
- 294 **3**  
 295     *M\_GA\_macro\_result\_decl(int; clock\_settime; time.h;;)*  
 296     **SEE:**        Assertion `GA_macro_result_decl` in §1.3.4  
 297     *Conformance for clock\_settime: PASS, NO\_OPTION*

- 298 **4**  
 299 *M\_GA\_macro\_args ( clock\_settime; time.h;;;)*  
 300 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 301 *Conformance for clock\_settime: PASS, NO\_OPTION*
- 302 **5**  
 303 *M\_GA\_stdC\_proto\_decl(int; clock\_gettime; clockid\_t clock\_id, struct timespec \*tp; time.h;;;)*  
 304 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 305 *Conformance for clock\_gettime: PASS[5, 6], NO\_OPTION*
- 306 **6**  
 307 *M\_GA\_commonC\_int\_result\_decl(clock\_gettime; , clockid\_t, clock\_id, struct timespec \*tp;*  
 308 *time.h;;;)*  
 309 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 310 *Conformance for clock\_gettime: PASS[5, 6], NO\_OPTION*
- 311 **7**  
 312 *M\_GA\_macro\_result\_decl(int; clock\_gettime; time.h;;;)*  
 313 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 314 *Conformance for clock\_gettime: PASS, NO\_OPTION*
- 315 **8**  
 316 *M\_GA\_macro\_args ( clock\_gettime; time.h;;;)*  
 317 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 318 *Conformance for clock\_gettime: PASS, NO\_OPTION*
- 319 **9**  
 320 *M\_GA\_stdC\_proto\_decl(int; clock\_getres; , clockid\_t clock\_id, struct timespec \*res; time.h;;;)*  
 321 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 322 *Conformance for clock\_getres: PASS[9, 10], NO\_OPTION*
- 323 **10**  
 324 *M\_GA\_commonC\_int\_result\_decl(clock\_getres; , clockid\_t clock\_id, struct timespec \*res;*  
 325 *time.h;;;)*  
 326 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 327 *Conformance for clock\_getres: PASS[9, 10], NO\_OPTION*
- 328 **11**  
 329 *M\_GA\_macro\_result\_decl(int; clock\_getres; time.h;;;)*  
 330 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 331 *Conformance for clock\_getres: PASS, NO\_OPTION*
- 332 **12**  
 333 *M\_GA\_macro\_args ( clock\_getres; time.h;;;)*  
 334 **SEE:** Assertion GA\_macro\_args in §2.7.3  
 335 *Conformance for clock\_getres: PASS, NO\_OPTION*
- 336 **1**  
 337 *M\_GA\_stdC\_proto\_decl(int; clock\_settime; clockid\_t clock\_id, const struct timespec \*tp; time.h;;;)*  
 338 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3  
 339 *Conformance for clock\_settime: PASS[1, 2], NO\_OPTION*
- 340 **2**  
 341 *M\_GA\_commonC\_int\_result\_decl(clock\_settime; time.h;;;)*  
 342 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 343 *Conformance for clock\_settime: PASS[1, 2], NO\_OPTION*
- 344 **3**  
 345 *M\_GA\_macro\_result\_decl(int; clock\_settime; time.h;;;)*  
 346 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4  
 347 *Conformance for clock\_settime: PASS, NO\_OPTION*

348 **4**  
349 *M\_GA\_macro\_args ( clock\_settime; time.h;;;)*  
350 **SEE:** Assertion GA\_macro\_args in §2.7.3  
351 *Conformance for clock\_settime: PASS, NO\_OPTION*

352 **14.2.1.2 Description**

353 **clock\_settime**  
354 **IF** *PCTS\_clock\_settime* **THEN**  
355 **IF** *PCTS\_GAP\_clock\_settime* and *PCTS\_DETECT\_EPERM\_clock\_settime* **THEN**  
356 **TEST:** A successful call to *clock\_settime (clock\_id, tp)* sets the specified clock,  
357 *clock\_id*, to the value specified by *tp*, and returns zero.  
358 **ELSE** *NO\_TEST\_SUPPORT*  
359 **ELSE** *NO\_OPTION*  
360 *Conformance for clock\_settime: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

361 **13** **IF** *PCTS\_clock\_settime* **THEN**  
362 **IF** *PCTS\_GAP\_clock\_settime* and *PCTS\_DETECT\_EPERM\_clock\_settime* **THEN**  
363 **TEST:** In a call to *clock\_settime()*, time values that are between two consecutive non-  
364 negative integer multiples of the resolution of the specified clock are truncated  
365 down to the smaller multiple of the resolution.  
366 **ELSE** *NO\_TEST\_SUPPORT*  
367 **ELSE** *NO\_OPTION*  
368 *Conformance for clock\_settime: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

369 **clock\_gettime**  
370 **IF** *PCTS\_clock\_gettime* **THEN**  
371 **TEST:** A successful call to *clock\_gettime (clock\_id, tp)* returns the current value *tp* for the  
372 specified clock, *clock\_id*.  
373 **ELSE** *NO\_OPTION*  
374 *Conformance for clock\_gettime: PASS, NO\_OPTION*

375 **clock\_getres**  
376 **IF** *PCTS\_clock\_getres* **THEN**  
377 **TEST:** A successful call to *clock\_getres (clock\_id, res)*, with a non\_NULL value of the  
378 argument *res*, stores the resolution of the clock specified by *clock\_id* into the location  
379 pointed to by *res*, and returns zero.  
380 **ELSE** *NO\_OPTION*  
381 *Conformance for clock\_getres: PASS, NO\_OPTION*

382 **14** **IF** *PCTS\_clock\_getres* **THEN**  
383 **TEST:** The PCD.1b documents clock resolutions, in §14.2.1.2.  
384 **ELSE** *NO\_OPTION*  
385 *Conformance for clock\_getres: PASS, NO\_OPTION*

386 **15** **TEST:** Clock resolutions cannot be set by a process.  
387 *Conformance for clock\_settime: PASS*

388 **16** **IF** *PCTS\_clock\_getres* **THEN**  
389 **TEST:** When the *res* argument to the call *clock\_getres ( , clock\_id, res)* is NULL, the clock  
390 resolution is not returned.  
391 **ELSE** *NO\_OPTION*  
392 *Conformance for clock\_getres: PASS, NO\_OPTION*

393 **17** **IF** *PCTS\_clock\_settime* **THEN**  
394 **IF** *PCTS\_\_clock\_getres* and *PCTS\_GAP\_clock\_settime* and *PCTS\_DETECT\_EPERM\_clock\_settime*  
395 **THEN**

396                   **TEST:**    When the time argument of *clock\_settime()* is not a multiple of the clock  
397    resolution, *res*, as determined by a call to *clock\_getres(, clock\_id, res)*, then the  
398    value is truncated to a multiple of *res*.  
399                   **ELSE NO\_TEST\_SUPPORT**  
400                   **ELSE NO\_OPTION**  
401                   *Conformance for clock\_settime: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

402   **D\_1 TEST:**    The PCD.1b documents whether the clocks are systemwide – that is, visible to all processes; or per-  
403    process – measuring time that is meaningful only within a process. in §14.2.1.2.  
404                   *Conformance for clock\_settime: PASS*

405   **18    SETUP:**    Include the header `<time.h>`.  
406                   **TEST:**    The constant `CLOCK_REALTIME` is defined.  
407                   *Conformance for clock\_settime: PASS*

408   **19    TEST:**    When the value `CLOCK_REALTIME` is used for *clock\_id*, the call *clock\_settime(clock\_id, tp)*  
409    sets the realtime clock for the system.  
410                   *Conformance for clock\_settime: PASS*

411   **20    TEST:**    When the value `CLOCK_REALTIME` is used for *clock\_id*, the call *clock\_gettime(clock\_id, tp)*  
412    interrogates the realtime clock for the system.  
413                   *Conformance for clock\_settime: PASS*

414   **21    TEST:**    When the value `CLOCK_REALTIME` is used for *clock\_id*, the call *clock\_get\_res(, clock\_id,*  
415    *res)* gets the resolution of the realtime clock for the system.  
416                   *Conformance for clock\_settime: PASS*

417   **22    IF PCTS\_clock\_gettime THEN**  
418                   **TEST:**    When the value `CLOCK_REALTIME` is used for *clock\_id*, the value returned by  
419    *clock\_gettime(clock\_id, tp)* represents the amount of time (in seconds and nanoseconds)  
420    since the Epoch.  
421                   **ELSE NO\_OPTION**  
422                   *Conformance for clock\_gettime: PASS, NO\_OPTION*

423   **23    IF PCTS\_clock\_settime THEN**  
424                    **IF PCTS\_GAP\_clock\_settime and PCTS\_DETECT\_EPERM\_clock\_settime THEN**  
425                    **TEST:**    When the value `CLOCK_REALTIME` is used for *clock\_id*, the value specified by  
426    *clock\_settime(clock\_id, tp)* represents the amount of time (in seconds and  
427    nanoseconds) since the Epoch.  
428                    **ELSE NO\_TEST\_SUPPORT**  
429                    **ELSE NO\_OPTION**  
430                    *Conformance for clock\_settime: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

431   **D\_1 IF a PCD.1b documents the following THEN**  
432                    **TEST:**    A PCD.1b that documents whether clocks in addition to the realtime clock are  
433    supported, and the interpretation of time values for any such clocks, does so in  
434    §14.2.1.2.  
435                    **ELSE NO\_OPTION**  
436                    *Conformance for clock\_settime: PASS, NO\_OPTION*

437   **24    IF PCTS\_clock\_settime THEN**  
438                    **TEST:**    The PCD.1b documents the effect of setting a clock via *clock\_settime()* on armed per-  
439    process times associated with that clock, in §14.2.1.2.  
440                    **ELSE NO\_OPTION**  
441                    *Conformance for clock\_settime: PASS, NO\_OPTION*

442   **25    IF PCTS\_clock\_settime THEN**  
443                    **TEST:**    The PCD.1b documents the appropriate privilege to set a particular clock, in  
444    §14.2.1.2.



445                   **ELSE NO\_OPTION**  
446                   *Conformance for clock\_settime: PASS, NO\_OPTION*

447 **D\_3 IF** *PCTS\_clock\_settime* and a PCD.1b documents the following **THEN**  
448                   **TEST:**     A PCD.1b that documents whether or not it supports the *clock\_settime()* function does  
449                                   so in §14.2.1.2.  
450                   **ELSE NO\_OPTION**  
451                   *Conformance for clock\_settime: PASS, NO\_OPTION*

452 **D\_4 IF** *PCTS\_clock\_gettime* and a PCD.1b documents the following **THEN**  
453                   **TEST:**     A PCD.1b that documents whether or not it supports the *clock\_gettime()* function does  
454                                   so in §14.2.1.2.  
455                   **ELSE NO\_OPTION**  
456                   *Conformance for clock\_gettime: PASS, NO\_OPTION*

457 **D\_5 IF** *PCTS\_clock\_getres* and a PCD.1b documents the following **THEN**  
458                   **TEST:**     A PCD.1b that documents whether or not it supports the *clock\_getres()* function does  
459                                   so in §14.2.1.2.  
460                   **ELSE NO\_OPTION**  
461                   *Conformance for clock\_getres: PASS, NO\_OPTION*

462 **14.2.1.3 Returns**

463 **R\_1 IF** *PCTS\_clock\_settime* **THEN**  
464                   **IF** *PCTS\_GAP\_clock\_settime* and *PCTS\_DETECT\_EPERM\_clock\_settime* **THEN**  
465                   **TEST:**     When a call to *clock\_settime()* completes successfully, the interface returns a value  
466                                   of 0.  
467                   **ELSE NO\_TEST\_SUPPORT**  
468                   **ELSE NO\_OPTION**  
469                   **SEE:**     Assertion *clock\_settime* in §14.2.1.4

470 **R\_2 IF** *PCTS\_clock\_settime* **THEN**  
471                   **IF** *PCTS\_GAP\_clock\_settime* and *PCTS\_DETECT\_EPERM\_clock\_settime* **THEN**  
472                   **TEST:**     When a call to *clock\_settime()* completes unsuccessfully, the interface returns a value  
473                                   of -1, and sets *errno* to indicate the error.  
474                                   *clock\_settime()*  
475                   **ELSE NO\_TEST\_SUPPORT**  
476                   **ELSE NO\_OPTION**  
477                   **SEE:**     All assertions in §14.2.1.4 controlled by *clock\_settime()*

478 **R\_3 IF** *PCTS\_clock\_gettime* **THEN**  
479                   **TEST:**     When a call to *clock\_gettime()* completes successfully, the interface returns a value  
480                                   of 0.  
481                   **ELSE NO\_OPTION**  
482                   **SEE:**     Assertion *clock\_gettime* in §14.2.1.4

483 **R\_4 IF** *PCTS\_clock\_gettime* **THEN**  
484                   **TEST:**     When a call to *clock\_gettime()* completes unsuccessfully, the interface returns a value  
485                                   of -1, and sets *errno* to indicate the error.  
486                                   *clock\_gettime()*  
487                   **ELSE NO\_OPTION**  
488                   **SEE:**     All assertions in §14.2.1.4 controlled by *clock\_gettime()*

489 **R\_5 IF** *PCTS\_clock\_getres* **THEN**  
490                   **TEST:**     When a call to *clock\_getres()* completes successfully, the interface returns a value of  
491                                   0.  
492                   **ELSE NO\_OPTION**  
493                   **SEE:**     Assertion *clock\_getres* in §14.2.1.4

494 **R\_6 IF PCTS\_clock\_getres THEN**  
 495       **TEST:** When a call to *clock\_gettime()* completes unsuccessfully, the interface returns a value  
 496               of -1, and sets *errno* to indicate the error.  
 497       **ELSE NO\_OPTION**  
 498       **SEE:** All assertions in §14.2.1.4 controlled by *clock\_getres()*

#### 499 14.2.1.4 Errors

500 **26 IF PCTS\_clock\_settime THEN**  
 501       **IF PCTS\_GAP\_clock\_settime and PCTS\_DETECT\_EPERM\_clock\_settime THEN**  
 502       **TEST:** A call to *clock\_settime (clock\_id, tp)* when the *clock\_id* argument does not specify a  
 503               known clock, returns a value of -1 and sets *errno* to [EINVAL].  
 504       **NOTE:** A subroutine is recommended that either returns an invalid clock name or indicates  
 505               that there is no way to generate an invalid clock name on the system.  
 506       **ELSE NO\_TEST\_SUPPORT**  
 507       **ELSE NO\_OPTION**  
 508       Conformance for *clock\_settime*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

509 **27 IF PCTS\_clock\_gettime THEN**  
 510       **TEST:** A call to *clock\_gettime (, clock\_id, tp)* when the *clock\_id* argument does not specify  
 511               a known clock, returns a value of -1 and sets *errno* to [EINVAL].  
 512       **NOTE:** A subroutine is recommended that either returns an invalid clock name or indicates  
 513               that there is no way to generate an invalid clock name on the system.  
 514       **ELSE NO\_OPTION**  
 515       Conformance for *clock\_gettime*: PASS, NO\_OPTION

516 **28 IF PCTS\_clock\_getres THEN**  
 517       **TEST:** A call to *clock\_getres (, clock\_id, res)* when the *clock\_id* argument does not specify  
 518               a known clock, returns a value of -1 and sets *errno* to [EINVAL].  
 519       **NOTE:** A subroutine is recommended that either returns an invalid clock name or indicates  
 520               that there is no way to generate an invalid clock name on the system.  
 521       **ELSE NO\_OPTION**  
 522       Conformance for *clock\_getres*: PASS, NO\_OPTION

523 **29 IF not PCTS\_clock\_gettime THEN**  
 524       **TEST:** A call to *clock\_settime()* returns a value of -1 and sets *errno* to [ENOSYS].  
 525       **ELSE NO\_OPTION**  
 526       Conformance for *clock\_settime*: PASS, NO\_OPTION

527 **30 IF not PCTS\_clock\_gettime THEN**  
 528       **TEST:** A call to *clock\_gettime()* returns a value of -1 and sets *errno* to [ENOSYS].  
 529       **ELSE NO\_OPTION**  
 530       Conformance for *clock\_gettime*: PASS, NO\_OPTION

531 **31 IF not PCTS\_clock\_getres THEN**  
 532       **TEST:** A call to *clock\_getres()* returns a value of -1 and sets *errno* to [ENOSYS].  
 533       **ELSE NO\_OPTION**  
 534       Conformance for *clock\_getres*: PASS, NO\_OPTION

535 **32 IF PCTS\_clock\_settime THEN**  
 536       **IF PCTS\_GAP\_clock\_settime and PCTS\_DETECT\_EPERM\_clock\_settime THEN**  
 537       **TEST:** A call to *clock\_settime (clock\_id, tp)* when the *tp* argument is outside the range  
 538               for te given clock id, returns a value of -1 and sets *errno* to [EINVAL].  
 539       **NOTE:** A subroutine is recommended that either returns a value outside the range value  
 540               for any given clock id or indicates that there is no way to generate a value  
 541               outside the range value for any given clock id on the system.  
 542       **ELSE NO\_TEST\_SUPPORT**

543           **ELSE NO\_OPTION**  
544           Conformance for `clock_settime`: `PASS, NO_TEST_SUPPORT, NO_OPTION`

545    **33**    **IF PCTS\_clock\_settime THEN**  
546            **IF PCTS\_GAP\_clock\_settime and PCTS\_DETECT\_EPERM\_clock\_settime THEN**  
547                **TEST:**    A call to `clock_settime (clock_id, tp)` when the `tp` argument specified a  
548                            nanosecond value less than zero, returns a value of -1 and sets `errno` to  
549                            [EINVAL].  
550                **ELSE NO\_TEST\_SUPPORT**  
551            **ELSE NO\_OPTION**  
552            Conformance for `clock_settime`: `PASS, NO_TEST_SUPPORT, NO_OPTION`

553    **34**    **IF PCTS\_clock\_settime THEN**  
554            **IF PCTS\_GAP\_clock\_settime and PCTS\_DETECT\_EPERM\_clock\_settime and {INT\_MAX} <=10e9**  
555            **THEN**  
556                **TEST:**    A call to `clock_settime (clock_id, tp)` when the `tp` argument specified a  
557                            nanosecond value greater than or equal to 1000 million, returns a value of -1  
558                            and sets `errno` to [EINVAL].  
559                **ELSE NO\_TEST\_SUPPORT**  
560            **ELSE NO\_OPTION**  
561            Conformance for `clock_settime`: `PASS, NO_TEST_SUPPORT, NO_OPTION`

562    **35**    **IF PCTS\_clock\_settime THEN**  
563            **IF PCTS\_RAP\_clock\_settime and PCTS\_DETECT\_EPERM\_clock\_settime THEN**  
564                **TEST:**    A call to `clock_settime()` when the requesting process does not have the  
565                            appropriate privilege to set the specified clock, returns a value of -1 and sets  
566                            `errno` to [EPERM].  
567                **ELSE NO\_TEST\_SUPPORT**  
568            **ELSE NO\_OPTION**  
569            Conformance for `clock_settime`: `PASS, NO_TEST_SUPPORT, NO_OPTION`

570    **14.2.2    Create a Per-Process Timer**

571    Function: `timer_create()`

572    **14.2.2.1 Synopsis**

573    **1**  
574            ***M\_GA\_stdC\_proto\_decl(int; timer\_create; clockid\_t clock\_id, struct sigevent \*evp, timer\_t \*timerid;***  
575            ***signal.h; time.h;)***  
576            **SEE:**    Assertion `GA_stdC_proto_decl` in §2.7.3  
577            Conformance for `timer_create`: `PASS[1, 2], NO_OPTION`

578    **2**  
579            ***M\_GA\_commonC\_int\_result\_decl(timer\_create; signal.h; time.h;)***  
580            **SEE:**    Assertion `GA_commonC_int_result_decl` in §2.7.3  
581            Conformance for `timer_create`: `PASS[1, 2], NO_OPTION`

582    **3**  
583            ***M\_GA\_macro\_result\_decl(int; timer\_create; signal.h; time.h;)***  
584            **SEE:**    Assertion `GA_macro_result_decl` in §1.3.4  
585            Conformance for `timer_create`: `PASS, NO_OPTION`

586    **4**  
587            ***M\_GA\_macro\_args ( timer\_create; signal.h; time.h;)***  
588            **SEE:**    Assertion `GA_macro_args` in §2.7.3  
589            Conformance for `timer_create`: `PASS, NO_OPTION`

590 **14.2.2.2 Description**591 **timer\_create**

- 592 **IF** *PCTS\_timer\_create* **THEN**  
 593     **TEST:**     A successful call to *timer\_create(clock\_id, evp, timerid)*, creates a per-process timer  
 594                    using the specified clock, *clock\_id*, as the timing base; sets the location referenced by  
 595                    *timerid*, to a timer ID of type *timer\_t* used to identify the timer in timer requests (see  
 596                    §14.2.4), and returns the value zero.  
 597     **ELSE NO\_OPTION**  
 598     Conformance for *timer\_create*: *PASS, NO\_OPTION*
- 599 **5**     **IF** *PCTS\_timer\_create* **THEN**  
 600         **TEST:**     Following a call to *timer\_create(clock\_id, evp, timerid)*, the timer ID, to which the  
 601                    argument *timerid* is set, is unique within the calling process until the timer is deleted.  
 602     **ELSE NO\_OPTION**  
 603     Conformance for *timer\_create*: *PASS, NO\_OPTION*
- 604 **6**     **TEST:**     In calls to *timer\_create(clock\_id, evp, timerid)*, all legal values of *clock\_id* are defined in  
 605                    <time.h>.  
 606     Conformance for *timer\_create*: *PASS*
- 607 **7**     **IF** *PCTS\_timer\_create* **THEN**  
 608         **TEST:**     The timer whose ID is returned is in a disarmed state upon return from *timer\_create()*.  
 609     **ELSE NO\_OPTION**  
 610     Conformance for *timer\_create*: *PASS, NO\_OPTION*
- 611 **8**     **IF** *PCTS\_timer\_create* **THEN**  
 612         **TEST:**     Following the call *timer\_create(clock\_id, evp, timerid)*, the *evp* argument, if non-  
 613                    NULL, points to a *sigevent* structure that defines the asynchronous notification that will  
 614                    occur when the timer expires.  
 615     **ELSE NO\_OPTION**  
 616     Conformance for *timer\_create*: *PASS, NO\_OPTION*
- 617 **9**     **IF** *PCTS\_timer\_create* **THEN**  
 618         **TEST:**     In the call *timer\_create(clock\_id, evp, timerid)*, the *evp* argument must be allocated  
 619                    by the application..  
 620     **ELSE NO\_OPTION**  
 621     Conformance for *timer\_create*: *PASS, NO\_OPTION*
- 622 **10**    **IF** *PCTS\_timer\_create* **THEN**  
 623         **SETUP:**    Include the header <time.h>.  
 624         **TEST:**     The constants SIGEV\_SIGNAL and SIGEV\_NONE are defined and have different values.  
 625     **ELSE NO\_OPTION**  
 626     Conformance for *timer\_create*: *PASS, NO\_OPTION*
- 627 **11**    **IF** *PCTS\_timer\_create* **THEN**  
 628         **TEST:**     Following the call *timer\_create(clock\_id, evp, timerid)*, if the *sigev\_notify* member  
 629                    of *evp* is SIGEV\_SIGNAL, then the structure contains the signal number and an  
 630                    application-specific data value to be sent to the process when the timer expires.  
 631     **ELSE NO\_OPTION**  
 632     Conformance for *timer\_create*: *PASS, NO\_OPTION*
- 633 **12**    **IF** *PCTS\_timer\_create* **THEN**  
 634         **TEST:**     Following the call *timer\_create(clock\_id, evp, timerid)*, if the *sigev\_notify* member  
 635                    is SIGEV\_NONE, no notification is sent.  
 636     **ELSE NO\_OPTION**  
 637     Conformance for *timer\_create*: *PASS, NO\_OPTION*
- 638 **13**    **IF** *PCTS\_timer\_create* **THEN**

639                   **TEST:**    The PCD.1b documents the behavior for any value of *sigev\_notify*, other than  
640                                   SIGEV\_NONE or SIGEV\_SIGNAL in §14.2.2.2.  
641                   **ELSE NO\_OPTION**  
642                   *Conformance for timer\_create: PASS, NO\_OPTION*

643   **14**           **IF** *PCTS\_timer\_create* **THEN**  
644                   **TEST:**    There is a set of clocks that can be used as timing bases for per-process timers.  
645                   **ELSE NO\_OPTION**  
646                   *Conformance for timer\_create: PASS, NO\_OPTION*

647   **15**           **IF** *PCTS\_timer\_create* **THEN**  
648                   **TEST:**    There is at least one mechanism for notifying the process of timer expiration events.  
649                   **ELSE NO\_OPTION**  
650                   *Conformance for timer\_create: PASS, NO\_OPTION*

651   **D\_1** **IF** *PCTS\_timer\_create* **THEN**  
652                   **TEST:**    The PCD.1b documents the set of clocks that can be used as timing bases for per-  
653                                   process timers and one or more mechanisms for notifying the process of timer  
654                                   expiration events, in §14.2.2.2.  
655                   **ELSE NO\_OPTION**  
656                   *Conformance for timer\_create: PASS, NO\_OPTION*

657   **16**           **IF** *PCTS\_timer\_create* **THEN**  
658                    **SETUP:**    Include the header <time.h>.  
659                    **TEST:**    The constant CLOCK\_REALTIME is defined.  
660                   **ELSE NO\_OPTION**  
661                   *Conformance for timer\_create: PASS, NO\_OPTION*

662   **17**           **IF** *PCTS\_timer\_create* **THEN**  
663                    **TEST:**    In a call to *timer\_create(clock\_id, evp, timerid)*, CLOCK\_REALTIME is a legitimate  
664                                   value for *clock\_id*.  
665                   **ELSE NO\_OPTION**  
666                   *Conformance for timer\_create: PASS, NO\_OPTION*

667   **18**           **IF** *PCTS\_timer\_create* **THEN**  
668                    **IF** {\_POSIX\_REALTIME\_SIGNALS} **THEN**  
669                      **SETUP:**    Call *timer\_create(clock\_id, evp, timerid)*, with *evp* argument pointing to a  
670                                   *sigevent* structure that specifies SIGEV\_SIGNAL, and SA\_SIGINFO set for the  
671                                   expiration signal.  
672                      **TEST:**    The signal and application-defined value specified in the *sigevent* structure are  
673                                   queued to the process on timer expiration.  
674                      **ELSE NO\_TEST\_SUPPORT**  
675                    **ELSE NO\_OPTION**  
676                    *Conformance for timer\_create: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

677   **19**           **IF** *PCTS\_timer\_create* **THEN**  
678                    **IF** {\_POSIX\_REALTIME\_SIGNALS} **THEN**  
679                      **SETUP:**    Call *timer\_create(clock\_id, evp, timerid)*, with *evp* argument set to NULL and  
680                                   SA\_SIGINFO set for the expiration signal.  
681                      **TEST:**    A default signal is queued to the process and the signal data value is set to the  
682                                   timer ID.  
683                      **ELSE NO\_TEST\_SUPPORT**  
684                    **ELSE NO\_OPTION**  
685                    *Conformance for timer\_create: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

686   **D\_2** **IF** *PCTS\_timer\_create* and a PCD.1b documents the following **THEN**  
687                    **TEST:**    A PCD.1b that documents whether the realtime signal is queued and what value, if any,  
688                                   is sent, if SA\_SIGINFO is not set for the expiration signal, does so in §14.2.2.2.

689           **ELSE NO\_OPTION**  
690           *Conformance for timer\_create: PASS, NO\_OPTION*

691   **20**   **IF PCTS\_timer\_create THEN**  
692           **IF** not {\_POSIX\_REALTIME\_SIGNALS} **THEN**  
693               **SETUP:** Call *timer\_create(clock\_id, evp, timerid)*, with *evp* pointing to a *sigevent*  
694               structure that specifies SIGEV\_SIGNAL.  
695               **TEST:** The signal number defined in the *sigevent* structure is sent to the process on  
696               timer expiration.  
697               **ELSE NO\_TEST\_SUPPORT**  
698           **ELSE NO\_OPTION**  
699           *Conformance for timer\_create: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

700   **21**   **IF PCTS\_timer\_create THEN**  
701           **IF** not {\_POSIX\_REALTIME\_SIGNALS} **THEN**  
702               **TEST:** Following the call *timer\_create(clock\_id, evp, timerid)*, if *evp* is NULL, then a  
703               default signal is sent to the process.  
704               **ELSE NO\_TEST\_SUPPORT**  
705           **ELSE NO\_OPTION**  
706           *Conformance for timer\_create: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

707   **22**   **IF PCTS\_timer\_create THEN**  
708               **SETUP:** Call *timer\_create(clock\_id, evp, timerid)*, with a *clock\_id* value of CLOCK\_REALTIME,  
709               and an *evp* of NULL.  
710               **TEST:** The default signal is SIGALRM.  
711           **ELSE NO\_OPTION**  
712           *Conformance for timer\_create: PASS, NO\_OPTION*

713   **23**   **IF PCTS\_timer\_create THEN**  
714               **TEST:** The PCD.1b documents the default signal number for any clock other than  
715               CLOCK\_REALTIME, in §14.2.2.2.  
716           **ELSE NO\_OPTION**  
717           *Conformance for timer\_create: PASS, NO\_OPTION*

718   **24**   **IF PCTS\_timer\_create THEN**  
719               **TEST:** Per-process timers are not inherited by a child process across a *fork()*.  
720           **ELSE NO\_OPTION**  
721           *Conformance for timer\_create: PASS, NO\_OPTION*

722   **25**   **FOR:** *execl()*, *execv()*, *execle()*, *execve()*, *exelp()*, and *execvp()*  
723           **IF PCTS\_timer\_create THEN**  
724               **TEST:** Per-process timers are disarmed and deleted by a successful call to *function()*.  
725               **NOTE:** The assertion is tested once for each function specified in the FOR clause. The  
726               assertion is to be read by substituting *function()* with the current function specified in  
727               the FOR clause. The name of the function also is to be substituted for each occurrence  
728               in the construct *PCTS\_function*.  
729           **ELSE NO\_OPTION**  
730           *Conformance for timer\_create: PASS, NO\_OPTION*

731   **D\_3** **IF PCTS\_timer\_create** and a PCD.1b documents the following **THEN**  
732               **TEST:** A PCD.1b that documents whether or not it supports the *timer\_create()* function does  
733               so in §14.2.2.2.  
734           **ELSE NO\_OPTION**  
735           *Conformance for timer\_create: PASS, NO\_OPTION*

736 **14.2.2.3 Returns**737 **R\_1 IF PCTS\_timer\_create THEN**

738 **TEST:** When a call to *timer\_create(clock\_id, evp, timerid)* completes successfully, the  
 739 interface returns a value of 0, and updates the location referenced by *timerid* to a  
 740 *timer\_t*, which can be passed to the per-process timer calls (see §14.2.4).

741 **ELSE NO\_OPTION**

742 **SEE:** Assertion *timer\_create* in §14.2.2.4

743 **R\_2 IF PCTS\_timer\_create THEN**

744 **TEST:** When a call to *timer\_create(clock\_id, evp, timerid)* completes unsuccessfully, the  
 745 interface returns a value of -1, and sets *errno* to indicate the error.

746 **ELSE NO\_OPTION**

747 **SEE:** All assertions in §14.2.2.4

748 **D\_4 IF PCTS\_timer\_create and A PCD.1b documents the following THEN**

749 **TEST:** A PCD.1b that documents the value of *timerid* if an error occurs, following a call to  
 750 *timer\_create(clock\_id, evp, timerid)*, does so in §14.2.2.4.

751 **ELSE NO\_OPTION**

752 *Conformance for timer\_create: PASS, NO\_OPTION*

753 **14.2.2.4 Errors**754 **26 IF PCTS\_timer\_create THEN**

755 **TEST:** A call to *timer\_create()*, when the system lacks sufficient signal queuing resources to  
 756 honor the request, returns a value of -1 and sets *errno* to [EAGAIN].

757 **NOTE:** The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to  
 758 write a portable test.

759 **ELSE NO\_OPTION**

760 *Conformance for timer\_create: PASS, NO\_OPTION*

761 **27 IF PCTS\_timer\_create THEN**

762 **IF {TIMER\_MAX} <= PCTS\_TIMER\_MAX THEN**

763 **TEST:** A call to *timer\_create()*, when the calling process has already created all of the  
 764 timers it is allowed by this implementation, returns a value of -1 and sets *errno*  
 765 to [EAGAIN].

766 **ELSE NO\_TEST\_SUPPORT**

767 **ELSE NO\_OPTION**

768 *Conformance for timer\_create: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

769 **28 IF PCTS\_timer\_create THEN**

770 **TEST:** A call to *timer\_create()*, when the specified clock ID is not defined, returns a value of  
 771 -1 and sets *errno* to [EINVAL].

772 **NOTE:** A subroutine is recommended that either returns a value outside the range value for  
 773 any given clock id or indicates that there is no way to generate a value outside the  
 774 range value for any given clock id on the system.

775 **ELSE NO\_OPTION**

776 *Conformance for timer\_create: PASS, NO\_OPTION*

777 **29 IF not PCTS\_timer\_create THEN**

778 **TEST:** A call to *timer\_create()* returns a value of -1 and sets *errno* to [ENOSYS].

779 **ELSE NO\_OPTION**

780 *Conformance for timer\_create: PASS, NO\_OPTION*

781 **14.2.3 Delete a Per-Process Timer**

782 Function: *timer\_delete()*

783 **14.2.3.1 Synopsis**784 **1**785 *M\_GA\_stdC\_proto\_decl(int; timer\_delete; timer\_t timerid; time.h;;)*786 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3787 *Conformance for timer\_delete: PASS[1, 2], NO\_OPTION*788 **2**789 *M\_GA\_commonC\_int\_result\_decl(timer\_delete; time.h;;)*790 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3791 *Conformance for timer\_delete: PASS[1, 2], NO\_OPTION*792 **3**793 *M\_GA\_macro\_result\_decl(int; timer\_delete; time.h;;)*794 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4795 *Conformance for timer\_delete: PASS, NO\_OPTION*796 **4**797 *M\_GA\_macro\_args ( timer\_delete; time.h;;)*798 **SEE:** Assertion GA\_macro\_args in §2.7.3799 *Conformance for timer\_delete: PASS, NO\_OPTION*800 **14.2.3.2 Description**801 **timer\_delete**802 **IF** *PCTS\_timer\_delete* **THEN**803 **IF** *PCTS\_timer\_create* **THEN**804 **TEST:** A successful call to *timer\_delete(timerid)* deletes the specified timer, *timerid*,  
805 previously created by the *timer\_create()* function, and returns the value zero.806 **ELSE** *NO\_TEST\_SUPPORT*807 **ELSE** *NO\_OPTION*808 *Conformance for timer\_delete: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*809 **5**810 **IF** *PCTS\_timer\_delete* **THEN**811 **TEST:** When the timer is armed when *timer\_delete()* is called, the behavior is as if the timer  
812 is automatically disarmed before removal.813 **ELSE** *NO\_OPTION**Conformance for timer\_delete: PASS, NO\_OPTION*814 **D\_1** **IF** *PCTS\_timer\_delete* and a PCD.1b documents the following **THEN**815 **TEST:** A PCD.1b that documents the disposition of pending signals for the deleted timer, does  
816 so in §14.2.3.2.817 **ELSE** *NO\_OPTION*818 *Conformance for timer\_delete: PASS, NO\_OPTION*819 **D\_2** **IF** *PCTS\_timer\_delete* and a PCD.1b documents the following **THEN**820 **TEST:** A PCD.1b that documents whether or not it supports the *timer\_delete()* function does  
821 so in §14.2.3.2.822 **ELSE** *NO\_OPTION*823 *Conformance for timer\_delete: PASS, NO\_OPTION*824 **14.2.3.3 Returns**825 **R\_1** **IF** *PCTS\_timer\_delete* **THEN**826 **TEST:** When a call to *timer\_delete()* completes successfully, the interface returns a value of  
827 0.828 **ELSE** *NO\_OPTION*829 **SEE:** Assertion *timer\_delete* in §14.2.3.4



830 **R\_2 IF** *PCTS\_timer\_delete* **THEN**  
 831       **TEST:**     When a call to *timer\_delete()* completes unsuccessfully, the interface returns a value  
 832                   of -1, and sets *errno* to indicate the error.  
 833       **ELSE NO\_OPTION**  
 834       **SEE:**       All assertions in §14.2.3.4

#### 835 14.2.3.4 Errors

836 **6 IF** *PCTS\_timer\_delete* **THEN**  
 837       **TEST:**     A call to *timer\_delete(timerid)*, when the timer ID specified by *timerid* is not a valid  
 838                   timer ID, returns a value of -1 and sets *errno* to [EINVAL].  
 839       **NOTE:**     A subroutine is recommended that either returns an invalid timer ID or indicates that  
 840                   there is no way to generate an invalid timer ID on the system.  
 841       **ELSE NO\_OPTION**  
 842       Conformance for *timer\_delete*: PASS, NO\_OPTION

843 **7 IF** not *PCTS\_timer\_delete* **THEN**  
 844       **TEST:**     A call to *timer\_delete()* returns a value of -1 and sets *errno* to [ENOSYS].  
 845       **ELSE NO\_OPTION**  
 846       Conformance for *timer\_delete*: PASS, NO\_OPTION

#### 847 14.2.4 Per-Process Timers

848 Functions: *timer\_settime()*, *timer\_gettime()*, *timer\_getoverrun()*

##### 849 14.2.4.1 Synopsis

850 **1**  
 851       *M\_GA\_stdC\_proto\_decl(int; timer\_settime; timer\_t timerid, int flags, const struct itimerspec \*value,*  
 852       *struct itimerspec \*ovalue; time.h;;;)*  
 853       **SEE:**       Assertion GA\_stdC\_proto\_decl in §2.7.3  
 854       Conformance for *timer\_settime*: PASS[1, 2], NO\_OPTION

855 **2**  
 856       *M\_GA\_commonC\_int\_result\_decl(timer\_settime; time.h;;;)*  
 857       **SEE:**       Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
 858       Conformance for *timer\_settime*: PASS[1, 2], NO\_OPTION

859 **3**  
 860       *M\_GA\_macro\_result\_decl(int; timer\_settime; time.h;;;)*  
 861       **SEE:**       Assertion GA\_macro\_result\_decl in §1.3.4  
 862       Conformance for *timer\_settime*: PASS, NO\_OPTION

863 **4**  
 864       *M\_GA\_macro\_args ( timer\_settime; time.h;;;)*  
 865       **SEE:**       Assertion GA\_macro\_args in §2.7.3  
 866       Conformance for *timer\_settime*: PASS, NO\_OPTION

867 **5**  
 868       *M\_GA\_stdC\_proto\_decl(int; timer\_gettime; timer\_t timerid, struct itimerspec \*value; time.h;;;)*  
 869       **SEE:**       Assertion GA\_stdC\_proto\_decl in §2.7.3  
 870       Conformance for *timer\_gettime*: PASS[5, 6], NO\_OPTION

871 **6**  
 872       *M\_GA\_commonC\_int\_result\_decl(timer\_gettime; timer\_t timerid, struct itimerspec \*value; time.h;;;)*  
 873       **SEE:**       Assertion GA\_commonC\_int\_result\_decl in §2.7.3

874 *Conformance for timer\_gettime: PASS[5, 6], NO\_OPTION*

875 **7**

876 *M\_GA\_macro\_result\_decl(int; timer\_gettime; time.h;;)*

877 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4

878 *Conformance for timer\_gettime: PASS, NO\_OPTION*

879 **8**

880 *M\_GA\_macro\_args ( timer\_gettime; time.h;;)*

881 **SEE:** Assertion GA\_macro\_args in §2.7.3

882 *Conformance for timer\_gettime: PASS, NO\_OPTION*

883 **9**

884 *M\_GA\_stdC\_proto\_decl(int; timer\_getoverrun; , timer\_t timerid; time.h;;)*

885 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3

886 *Conformance for timer\_getoverrun: PASS[9, 10], NO\_OPTION*

887 **10**

888 *M\_GA\_commonC\_int\_result\_decl(timer\_getoverrun; timer\_t timerid; time.h;;)*

889 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3

890 *Conformance for timer\_getoverrun: PASS[9, 10], NO\_OPTION*

891 **11**

892 *M\_GA\_macro\_result\_decl(int; timer\_getoverrun; time.h;;)*

893 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4

894 *Conformance for timer\_getoverrun: PASS, NO\_OPTION*

895 **12**

896 *M\_GA\_macro\_args ( timer\_getoverrun; time.h;;)*

897 **SEE:** Assertion GA\_macro\_args in §2.7.3

898 *Conformance for timer\_getoverrun: PASS, NO\_OPTION*

899 **14.2.4.2 Description**

900 **timer\_settime**

901 **IF** PCTS\_timer\_settime **THEN**

902 **TEST:** A successful call to *timer\_settime(timerid, flags, value, ovalue)*, sets the time until the

903 next expiration of the timer specified by *timerid* from the *it\_value* member of the

904 *value* is nonzero, and returns the value zero.

905 **ELSE NO\_OPTION**

906 *Conformance for timer\_settime: PASS, NO\_OPTION*

907 **13** **IF** PCTS\_timer\_settime **THEN**

908 **TEST:** When the specified timer is already armed when *timer\_settime(timerid, flags, value,*

909 *ovalue)* is called, this call resets the time until next expiration to the *value* specified.

910 **ELSE NO\_OPTION**

911 *Conformance for timer\_settime: PASS, NO\_OPTION*

912 **14** **IF** PCTS\_timer\_settime **THEN**

913 **TEST:** In a successful call to *timer\_settime(timerid, flags, value, ovalue)*, if the *it\_value*

914 member of *value* is zero, the timer is disarmed.

915 **ELSE NO\_OPTION**

916 *Conformance for timer\_settime: PASS, NO\_OPTION*

917 **D\_1** **IF** PCTS\_timer\_settime and a PCD.1b documents the following **THEN**

918 **TEST:** A PCD.1b that documents the effect of disarming or resetting a timer on pending

919 expiration notifications, does so in §14.2.4.2.

920 **ELSE NO\_OPTION**

921 *Conformance for timer\_settime: PASS, NO\_OPTION*

- 922     **15**     **IF** *PCTS\_timer\_settime* **THEN**  
923             **SETUP:**    Include the header `<timer.h>`.  
924             **TEST:**     The constant `TIMER_ABSTIME` is defined.  
925     **ELSE NO\_OPTION**  
926     *Conformance for timer\_settime: PASS, NO\_OPTION*
- 927     **16**     **IF** *PCTS\_timer\_settime* **THEN**  
928             **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 (see POSIX.1b {3} §14.1.1) from when the call is made.  
929  
930  
931  
932     **ELSE NO\_OPTION**  
933     *Conformance for timer\_settime: PASS, NO\_OPTION*
- 934     **17**     **IF** *PCTS\_timer\_settime* **THEN**  
935             **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 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*.  
936  
937  
938  
939  
940     **ELSE NO\_OPTION**  
941     *Conformance for timer\_settime: PASS, NO\_OPTION*
- 942     **18**     **IF** *PCTS\_timer\_settime* **THEN**  
943             **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.  
944  
945  
946     **ELSE NO\_OPTION**  
947     *Conformance for timer\_settime: PASS, NO\_OPTION*
- 948     **19**     **IF** *PCTS\_timer\_settime* **THEN**  
949             **TEST:**     A successful call to *timer\_settime(timerid, flags, value, ovalue)*, with a zero value of *it\_interval* specifies a nonperiodic (one-time) timer.  
950  
951     **ELSE NO\_OPTION**  
952     *Conformance for timer\_settime: PASS, NO\_OPTION*
- 953     **20**     **IF** *PCTS\_timer\_settime* **THEN**  
954             **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*.  
955  
956  
957     **ELSE NO\_OPTION**  
958     *Conformance for timer\_settime: PASS, NO\_OPTION*
- 959     **21**     **IF** *PCTS\_timer\_settime* **THEN**  
960             **TEST:**     In calls to *timer\_settime()*, time values that are between two consecutive nonnegative integer multiples of the resolution of the specified timer are rounded up to the larger multiple of the resolution; quantization error does not cause the timer to expire earlier than the rounded time value.  
961  
962  
963  
964     **ELSE NO\_OPTION**  
965     *Conformance for timer\_settime: PASS, NO\_OPTION*
- 966     **22**     **IF** *PCTS\_timer\_settime* **THEN**  
967             **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 together with the previous timer reload value.  
968  
969  
970  
971     **ELSE NO\_OPTION**  
972     *Conformance for timer\_settime: PASS, NO\_OPTION*

973 **23** **IF** *PCTS\_timer\_settime* **THEN**  
974 **TEST:** When the timer *timerid* is disarmed and the argument *ovalue* is not **NULL**, the call  
975 *timer\_settime(timerid, flags, value, ovalue)* stores, in the location referenced by  
976 *ovalue*, a value representing zero, together with the previous timer reload value.  
977 **ELSE NO\_OPTION**  
978 *Conformance for timer\_settime: PASS, NO\_OPTION*

979 **24** **IF** *PCTS\_timer\_settime* **THEN**  
980 **IF** *PCTS\_timer\_gettime* **THEN**  
981 **TEST:** In the call *timer\_settime(timerid, flags, value, ovalue)*, the members of *ovalue*  
982 are subject to the resolution of the timer, and they are the same values that  
983 would be returned by a *timer\_gettime()* call at the point in the time.  
984 **NOTE:** There is no known portable test method for this assertion.  
985 **ELSE NO\_TEST\_SUPPORT**  
986 **ELSE NO\_OPTION**  
987 *Conformance for timer\_settime: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

988 **timer\_gettime**  
989 **IF** *PCTS\_timer\_gettime* **THEN**  
990 **TEST:** A successful call to *timer\_gettime()* stores both the reload value of the timer and the  
991 amount of time until the specified timer, *timerid*, expires into the space pointed to by  
992 the *value* argument; and returns the value zero.

993 The *it\_value* member of this structure contains the amount of time before the timer  
994 expires, or zero if the timer is disarmed.

995 The *it\_interval* member of *value* contains the reload value last set by *timer\_settime()*.  
996 **ELSE NO\_OPTION**  
997 *Conformance for timer\_gettime: PASS, NO\_OPTION*

998 **25** **IF** *PCTS\_timer\_gettime* **THEN**  
999 **TEST:** The amount of time before the timer expires is returned as the interval until timer  
1000 expiration, even if the timer is armed with absolute time.  
1001 **ELSE NO\_OPTION**  
1002 *Conformance for timer\_gettime: PASS, NO\_OPTION*

1003 **26** **IF** **{\_POSIX\_REALTIME\_SIGNALS}** **THEN**  
1004 **TEST:** Only a single signal is queued to the process for a given timer at any point in time.  
1005 **ELSE NO\_TEST\_SUPPORT**  
1006 *Conformance for timer\_settime: PASS, NO\_TEST\_SUPPORT*

1007 **27** **IF** **{\_POSIX\_REALTIME\_SIGNALS}** **THEN**  
1008 **TEST:** When a timer for which a signal is still pending expires, no signal is queued, a timer  
1009 overrun occurs.  
1010 **ELSE NO\_TEST\_SUPPORT**  
1011 *Conformance for timer\_settime: PASS, NO\_TEST\_SUPPORT*

1012 **timer\_getoverrun**  
1013 **IF** *PCTS\_timer\_getoverrun* **THEN**  
1014 **IF** **{\_POSIX\_REALTIME\_SIGNALS}** **THEN**  
1015 **TEST:** When a timer expiration signal is delivered to a process, the *timer\_getoverrun()*  
1016 function returns the timer expiration overrun count for the specified timer.  
1017 **NOTE:** There is no known portable test method for this assertion.  
1018 **ELSE NO\_TEST\_SUPPORT**  
1019 **ELSE NO\_OPTION**  
1020 *Conformance for timer\_getoverrun: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1021 **28** **IF** *PCTS\_timer\_getoverrun* **THEN**

1022           **TEST:**     The PCD.1b documents the value of {DELAYTIMER\_MAX}, in §14.2.4.2.  
1023           **ELSE NO\_OPTION**  
1024           *Conformance for timer\_getoverrun: PASS, NO\_OPTION*

1025   **29**       **IF PCTS\_timer\_getoverrun THEN**  
1026           **TEST:**     The overrun count returned by *timer\_getoverrun()* contains the number of extra timer  
1027                            expirations that occurred between the time the signal was generated (queued) and  
1028                            when it was delivered.  
1029           **NOTE:**     There is no known portable test method for this assertion.  
1030           **ELSE NO\_OPTION**  
1031           *Conformance for timer\_getoverrun: PASS, NO\_TEST, NO\_OPTION*

1032   **30**       **IF PCTS\_timer\_getoverrun THEN**  
1033           **IF** {DELAYTIMER\_MAX} <= PCTS\_DELAYTIMER\_MAX **THEN**  
1034           **TEST:**     When the number of extra expirations is greater than or equal to  
1035                            {DELAYTIMER\_MAX}, then *timer\_getoverrun()* returns {DELAYTIMER\_MAX}.  
1036           **NOTE:**     There is no known portable test method for this assertion.  
1037           **ELSE NO\_TEST\_SUPPORT**  
1038           **ELSE NO\_OPTION**  
1039           *Conformance for timer\_getoverrun: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

1040   **31**       **IF PCTS\_timer\_getoverrun THEN**  
1041           **TEST:**     The value returned by *timer\_getoverrun()* applies to the most recent expiration signal  
1042                            delivery for the timer.  
1043           **NOTE:**     There is no known portable test method for this assertion.  
1044           **ELSE NO\_OPTION**  
1045           *Conformance for timer\_getoverrun: PASS, NO\_TEST, NO\_OPTION*

1046   **D\_2** **IF PCTS\_timer\_getoverrun** and a PCD.1b documents the following **THEN**  
1047           **TEST:**     A PCD.1b that documents the meaning of the overrun count returned, if no expiration  
1048                            signal has been delivered for the timer, or if {\_POSIX\_REALTIME\_SIGNALS} is not  
1049                            supported, does so in §14.2.4.2.  
1050           **NOTE:**     There is no known portable test method for this assertion.  
1051           **ELSE NO\_OPTION**  
1052           *Conformance for timer\_getoverrun: PASS, NO\_OPTION*

1053   **D\_3** **IF PCTS\_timer\_settime** and a PCD.1b documents the following **THEN**  
1054           **TEST:**     A PCD.1b that documents whether or not it supports the *timer\_settime()* function does  
1055                            so in §14.2.4.2.  
1056           **ELSE NO\_OPTION**  
1057           *Conformance for timer\_settime: PASS, NO\_OPTION*

1058   **D\_4** **IF PCTS\_timer\_gettime** and a PCD.1b documents the following **THEN**  
1059           **TEST:**     A PCD.1b that documents whether or not it supports the *timer\_gettime()* function does  
1060                            so in §14.2.4.2.  
1061           **ELSE NO\_OPTION**  
1062           *Conformance for timer\_gettime: PASS, NO\_OPTION*

1063   **D\_5** **IF PCTS\_timer\_getoverrun** and a PCD.1b documents the following **THEN**  
1064           **TEST:**     A PCD.1b that documents whether or not it supports the *timer\_getoverrun()* function  
1065                            does so in §14.2.4.2.  
1066           **ELSE NO\_OPTION**  
1067           *Conformance for timer\_getoverrun: PASS, NO\_OPTION*

1068   **14.2.4.3 Returns**

1069   **R\_1** **IF PCTS\_timer\_settime THEN**

1070                   **TEST:**    When a call to *timer\_settime()* completes successfully, the interface returns a value of  
 1071                                    0.  
 1072                   **ELSE NO\_OPTION**  
 1073                   **SEE:**     Assertion *timer\_settime* in §14.2.4.4

1074 **R\_2 IF PCTS\_timer\_gettime THEN**  
 1075                   **TEST:**    When a call to *timer\_gettime()* completes successfully, the interface returns a value  
 1076                                    of 0.  
 1077                   **ELSE NO\_OPTION**  
 1078                   **SEE:**     Assertion *timer\_gettime* in §14.2.4.4

1079 **R\_3 IF PCTS\_timer\_settime THEN**  
 1080                   **TEST:**    When a call to *timer\_settime()* completes unsuccessfully, the interface returns a value  
 1081                                    of -1, and sets *errno* to indicate the error.  
 1082                                    *timer\_settime()*  
 1083                   **ELSE NO\_OPTION**  
 1084                   **SEE:**     All assertions in §14.2.4.4 controlled by *timer\_settime()*

1085 **R\_4 IF PCTS\_timer\_gettime THEN**  
 1086                   **TEST:**    When a call to *timer\_gettime()* completes unsuccessfully, the interface returns a value  
 1087                                    of -1, and sets *errno* to indicate the error.  
 1088                                    *timer\_gettime()*  
 1089                   **ELSE NO\_OPTION**  
 1090                   **SEE:**     All assertions in §14.2.4.4 controlled by *timer\_gettime()*

1091 **R\_5 IF PCTS\_timer\_getoverrun THEN**  
 1092                   **TEST:**    When a call to *timer\_getoverrun()* completes successfully, the interface returns the  
 1093                                    timer expiration overrun count as explained in POSIX.1b {3} §14.2.4.2.  
 1094                   **ELSE NO\_OPTION**  
 1095                   **SEE:**     Assertion *timer\_getoverrun* in §14.2.4.4

#### 1096 14.2.4.4 Errors

1097 **32 IF PCTS\_timer\_settime THEN**  
 1098                   **IF PCTS\_timer\_create and PCTS\_timer\_delete THEN**  
 1099                   **TEST:**    A call to *timer\_settime()*, when the *timerid* argument does not correspond to an  
 1100                                    ID returned by *timer\_create()* but not yet deleted by *timer\_delete()*, returns a  
 1101                                    value of -1 and sets *errno* to [EINVAL].  
 1102                   **NOTE:**    A subroutine is recommended that either returns an invalid timer ID or indicates  
 1103                                    that there is no way to generate an invalid timer ID on the system.  
 1104                   **ELSE NO\_TEST\_SUPPORT**  
 1105                   **ELSE NO\_OPTION**  
 1106                   Conformance for *timer\_settime*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1107 **33 IF PCTS\_timer\_gettime THEN**  
 1108                   **IF PCTS\_timer\_create and PCTS\_timer\_delete THEN**  
 1109                   **TEST:**    A call to *timer\_gettime()*, when the *timerid* argument does not correspond to an  
 1110                                    ID returned by *timer\_create()* but not yet deleted by *timer\_delete()*, returns a  
 1111                                    value of -1 and sets *errno* to [EINVAL].  
 1112                   **NOTE:**    A subroutine is recommended that either returns an invalid timer ID or indicates  
 1113                                    that there is no way to generate an invalid timer ID on the system.  
 1114                   **ELSE NO\_TEST\_SUPPORT**  
 1115                   **ELSE NO\_OPTION**  
 1116                   Conformance for *timer\_gettime*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1117 **34 IF PCTS\_timer\_getoverrun THEN**  
 1118                   **IF PCTS\_timer\_create and PCTS\_timer\_delete THEN**

1119                   **TEST:**     A call to *timer\_getoverrun*(, *timerid*) when the *timerid* argument does not  
1120                   correspond to an ID returned by *timer\_create*() but not yet deleted by  
1121                   *timer\_delete*(), returns a value of -1 and sets *errno* to [EINVAL].  
1122                   **NOTE:**     A subroutine is recommended that either returns an invalid timer ID or indicates  
1123                   that there is no way to generate an invalid timer ID on the system.  
1124                   **ELSE NO\_TEST\_SUPPORT**  
1125                   **ELSE NO\_OPTION**  
1126                   Conformance for *timer\_getoverrun*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1127   **35**           **IF** not *PCTS\_timer\_settime* **THEN**  
1128                   **TEST:**     A call to *timer\_settime*() returns a value of -1 and sets *errno* to [ENOSYS].  
1129                   **ELSE NO\_OPTION**  
1130                   Conformance for *timer\_settime*: *PASS, NO\_OPTION*

1131   **36**           **IF** not *PCTS\_timer\_gettime* **THEN**  
1132                   **TEST:**     A call to *timer\_gettime*() returns a value of -1 and sets *errno* to [ENOSYS].  
1133                   **ELSE NO\_OPTION**  
1134                   Conformance for *timer\_gettime*: *PASS, NO\_OPTION*

1135   **37**           **IF** not *PCTS\_timer\_getoverrun* **THEN**  
1136                   **TEST:**     A call to *timer\_getoverrun* returns a value of -1 and sets *errno* to [ENOSYS].  
1137                   **ELSE NO\_OPTION**  
1138                   Conformance for *timer\_getoverrun*: *PASS, NO\_OPTION*

1139   **38**           **IF** *PCTS\_timer\_settime* **THEN**  
1140                   **TEST:**     A call to *timer\_settime*(*timerid*, *flags*, *value*, *ovalue*), when the *value* structure  
1141                   specifies a nanosecond value less than zero, returns a value of -1 and sets *errno* to  
1142                   [EINVAL].  
1143                   **ELSE NO\_OPTION**  
1144                   Conformance for *timer\_settime*: *PASS, NO\_OPTION*

1145   **39**           **IF** *PCTS\_timer\_settime* **THEN**  
1146                   **IF** {INT\_MAX}<= 10e9 **THEN**  
1147                   **TEST:**     A call to *timer\_settime*(*timerid*, *flags*, *value*, *ovalue*), when the *value* structure  
1148                   specifies a nanosecond greater than or equal to 1000 million, returns a value  
1149                   of -1 and sets *errno* to [EINVAL].  
1150                   **ELSE NO\_TEST\_SUPPORT**  
1151                   **ELSE NO\_OPTION**  
1152                   Conformance for *timer\_settime*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

## 1153   **14.2.5   High Resolution Sleep**

1154   Function: *nanosleep*()

### 1155   **14.2.5.1 Synopsis**

1156   **1**  
1157                   *M\_GA\_stdC\_proto\_decl(int; nanosleep; const struct timespec \*rmt; time.h;;)*  
1158                   **SEE:**     Assertion GA\_stdC\_proto\_decl in §2.7.3  
1159                   Conformance for *nanosleep*: *PASS[1, 2], NO\_OPTION*

1160   **2**  
1161                   *M\_GA\_commonC\_int\_result\_decl(nanosleep; time.h;;)*  
1162                   **SEE:**     Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
1163                   Conformance for *nanosleep*: *PASS[1, 2], NO\_OPTION*

1164   **3**  
1165                   *M\_GA\_macro\_result\_decl(int; nanosleep; time.h;;)*

1166           **SEE:**       Assertion GA\_macro\_result\_decl in §1.3.4  
 1167           *Conformance for nanosleep: PASS, NO\_OPTION*

1168       **4**  
 1169           *M\_GA\_macro\_args ( nanosleep; time.h;;;)*  
 1170           **SEE:**       Assertion GA\_macro\_args in §2.7.3  
 1171           *Conformance for nanosleep: PASS, NO\_OPTION*

1172       **14.2.5.2 Description**

1173       **nanosleep**

1174           **IF** PCTS\_nanosleep **THEN**  
 1175               **TEST:**     A successful call to *nanosleep(rqtp, rmtp)* causes the current process to be suspended  
 1176                           from execution until the time interval specified by the *rqtp* argument has elapsed,  
 1177                           whereupon it returns zero.  
 1178           **ELSE** NO\_OPTION  
 1179           *Conformance for nanosleep: PASS, NO\_OPTION*

1180       **5**           **IF** PCTS\_nanosleep **THEN**  
 1181               **TEST:**     In a successful call to *nanosleep(rqtp, rmtp)*, the suspension time is not less than the  
 1182                           time specified by the *rqtp*, as measured by the system clock, CLOCK\_REALTIME.  
 1183           **ELSE** NO\_OPTION  
 1184           *Conformance for nanosleep: PASS, NO\_OPTION*

1185       **6**           **IF** PCTS\_nanosleep **THEN**  
 1186               **TEST:**     In a successful call to *nanosleep(rqtp, rmtp)*, the suspension time may be longer than  
 1187                           requested because the argument value is rounded up to an integer multiple of the sleep  
 1188                           resolution.  
 1189               **NOTE:**     There is no known portable test method for this assertion.  
 1190           **ELSE** NO\_OPTION  
 1191           *Conformance for nanosleep: PASS, NO\_TEST, NO\_OPTION*

1192       **7**           **IF** PCTS\_nanosleep **THEN**  
 1193               **TEST:**     In a successful call to *nanosleep(rqtp, rmtp)*, the suspension time may be longer than  
 1194                           requested because of the scheduling of other activity by the system.  
 1195               **NOTE:**     There is no known portable test method for this assertion.  
 1196           **ELSE** NO\_OPTION  
 1197           *Conformance for nanosleep: PASS, NO\_TEST, NO\_OPTION*

1198       **8**           **IF** PCTS\_nanosleep **THEN**  
 1199               **TEST:**  
 1200           **ELSE** NO\_OPTION  
 1201           *Conformance for nanosleep: PASS, NO\_OPTION*

1202       **9**           **IF** PCTS\_nanosleep **THEN**  
 1203               **TEST:**     The use of the *nanosleep()* function has no effect on the action or blockage of any  
 1204                           signal.  
 1205           **ELSE** NO\_OPTION  
 1206           *Conformance for nanosleep: PASS, NO\_OPTION*

1207       **D\_1** **IF** PCTS\_nanosleep and a PCD.1b documents the following **THEN**  
 1208               **TEST:**     A PCD.1b that documents whether or not it supports the *nanosleep()* function does so  
 1209                           in §14.2.5.2.  
 1210           **ELSE** NO\_OPTION  
 1211           *Conformance for nanosleep: PASS, NO\_OPTION*

1212       **14.2.5.3 Returns**



1213 **R\_1 IF PCTS\_nanosleep THEN**  
1214       **TEST:** When a call to *nanosleep()* returns because the requested time has elapsed, the  
1215                   interface returns a value of 0.  
1216       **ELSE NO\_OPTION**  
1217       **SEE:** Assertion *nanosleep* in §14.2.5.4

1218 **R\_2 IF PCTS\_nanosleep THEN**  
1219       **TEST:** When a call to *nanosleep()* returns because it has been interrupted by a signal, the  
1220                   interface returns a value of -1, and sets *errno* to indicate the interruption.  
1221       **ELSE NO\_OPTION**  
1222       **SEE:** All assertions in §14.2.5.4

1223 **10 IF PCTS\_nanosleep THEN**  
1224       **TEST:** When the *rmtpt* argument is non-NULL, the *timespec* structure referenced by it is  
1225                   updated by *nanosleep(rqtp, rmtpt)* to contain the amount of time remaining in the  
1226                   interval (the requested time minus the time actually slept).  
1227       **ELSE NO\_OPTION**  
1228       Conformance for *nanosleep*: PASS, NO\_OPTION

1229 **11 IF PCTS\_nanosleep THEN**  
1230       **TEST:** When the *rmtpt* argument is NULL, the remaining time is not returned by  
1231                   *nanosleep(rqtp, rmtpt)*.  
1232       **ELSE NO\_OPTION**  
1233       Conformance for *nanosleep*: PASS, NO\_OPTION

1234 **R\_3 IF PCTS\_nanosleep THEN**  
1235       **TEST:** When a call to *nanosleep()* completes unsuccessfully, the interface returns a value of  
1236                   -1, and sets *errno* to indicate the error.  
1237       **ELSE NO\_OPTION**  
1238       **SEE:** All assertions in §14.2.5.4

1239 **14.2.5.4 Errors**

1240 **12 IF PCTS\_nanosleep THEN**  
1241       **TEST:** A call to *nanosleep()*, when the *nanosleep()* function is interrupted by a signal, returns  
1242                   a value of -1 and sets *errno* to [EINTR].  
1243       **ELSE NO\_OPTION**  
1244       Conformance for *nanosleep*: PASS, NO\_OPTION

1245 **13 IF PCTS\_nanosleep THEN**  
1246       **TEST:** A call to *nanosleep(rqtp, rmtpt)*, when the *rqtp* argument specifies a nanosecond value  
1247                   less than zero, returns a value -1 and sets *errno* to [EINVAL].  
1248       **ELSE NO\_OPTION**  
1249       Conformance for *nanosleep*: PASS, NO\_OPTION

1250 **14 IF PCTS\_nanosleep THEN**  
1251       **IF INT\_MAX<=10e9 THEN**  
1252           **TEST:** A call to *nanosleep(rqtp, rmtpt)*, when the *rqtp* argument specifies a nanosecond  
1253                   value greater than or equal to 1000 million, returns a value of -1 and sets *errno*  
1254                   to [EINVAL].  
1255           **ELSE NO\_TEST\_SUPPORT**  
1256       **ELSE NO\_OPTION**  
1257       Conformance for *nanosleep*: PASS, NO\_TEST\_SUPPORT, NO\_OPTION

1258 **15 IF not PCTS\_nanosleep THEN**  
1259       **TEST:** A call to *nanosleep()* returns a value of -1 and sets *errno* to [ENOSYS].  
1260       **ELSE NO\_OPTION**  
1261       Conformance for *nanosleep*: PASS, NO\_OPTION



This page is intentionally blank.

## Section 15: Message Passing

### 180 15.1 Data Definitions for Message Queues

181 **D\_1 SETUP:** Include the header <MQUEUE.H>.

182 **TEST:** A PCD.1b that documents the symbols allowed by this standard to be in the headers  
183 <SYS/TYPES.H> <FCNT1.H> <TIME.H> <SIGNAL.H> are visible does so in §15.1.

184 **ELSE NO\_OPTION**

185 *Conformance for mq\_intro: PASS, NO\_OPTION*

#### 186 15.1.1 Data Structures

187 **1 SETUP:** Include the header <mqueue.h>.

188 **TEST:** The types *mqd\_t* and *struct sigevent* are defined.

189 *Conformance for mq\_hdr: PASS*

190 **D\_1 TEST:** The PCD.1b documents the definition of *mqd\_t* and *struct sigevent*, in §15.1.1.

191 *Conformance for mq\_hdr: PASS*

192 **3 SETUP:** Include the header <mqueue.h>.

193 **TEST:** The structure *mq\_attr* is defined and has at least the following members.

194	Member	Member	
195	Type	Name	Description
196	<i>long</i>	<i>mq_flags</i>	Message queue flags.
197	<i>long</i>	<i>mq_maxmsg</i>	Maximum number of messages.
198	<i>long</i>	<i>mq_msgsize</i>	Maximum message size.
199	<i>long</i>	<i>mq_curmsgs</i>	Number of messages currently queued.

200 *Conformance for mq\_hdr: PASS*

201 **D\_2 IF** a PCD.1b documents the following **THEN**

202 **TEST:** A PCD.1b that documents extensions to *mq\_attr*, as permitted in POSIX.1b{3} §1.3.1.1  
203 item (2), does so in §15.1.1.

204 **ELSE NO\_OPTION**

205 *Conformance for mq\_hdr: PASS, NO\_OPTION*

206 **4 SETUP:** Include the header <mqueue.h>.

207 **TEST:** Extensions to *mq\_attr* that may change the behavior of the application with respect to this  
208 standard when those fields in the structure are uninitialized, are enabled as required by  
209 POSIX.1b {3} §1.3.1.1.

210 **NOTE:** The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write  
211 a portable test.

212 *Conformance for mq\_hdr: PASS, NO\_TEST*

213 **D\_3** **IF** a PCD.1b documents the following **THEN**  
 214       **TEST:** A PCD.1b that documents the existence of flags other than O\_NONBLOCK, does so in  
 215                   §15.1.1.  
 216       **ELSE NO\_OPTION**  
 217       Conformance for *mq\_hdr*: *PASS, NO\_OPTION*

218 **M\_GA\_mqOpenMaxFD** () =  
 219       **IF** *PCTS\_MQ\_FILE\_DESCRIPTOR* **THEN**  
 220           **IF** ({*OPEN\_MAX*} <= *PCTS\_OPEN\_MAX*) and *PCTS\_mq\_open* **THEN**  
 221               **TEST:** A process calling *mq\_open*() can simultaneously open a combination of files and  
 222                   message queues totaling at least {*OPEN\_MAX*}.  
 223               **TR:** Test for opening {*OPEN\_MAX*} message queues.  
 224                                   Test for opening a message queue after opening {*OPEN\_MAX*}-1 files.  
 225                                   Test for opening a file after opening {*OPEN\_MAX*}-1 message queues.  
 226               **ELSE NO\_TEST\_SUPPORT**  
 227       **ELSE NO\_OPTION**

228 **GA\_mqOpenMaxFD**  
 229       **FOR:** *mq\_open*()  
 230       **M\_GA\_mqOpenMaxFD**()  
 231       Conformance for *mq\_hdr*: *PASS, NO\_OPTION*

232 **M\_GA\_mqPCTSOpenMaxFD** () =  
 233       **IF** *PCTS\_MQ\_FILE\_DESCRIPTOR* **THEN**  
 234           **IF** ({*OPEN\_MAX*} > *PCTS\_OPEN\_MAX*) and *PCTS\_mq\_open* **THEN**  
 235               **TEST:** A process calling *mq\_open*() can simultaneously open a combination of files and  
 236                   message queues totaling at least *PCTS\_OPEN\_MAX*.  
 237               **TR:** Test for opening *PCTS\_OPEN\_MAX* message queues.  
 238                                   Test for opening a message queue after opening *PCTS\_OPEN\_MAX*-1 files.  
 239                                   Test for opening a file after opening *PCTS\_OPEN\_MAX*-1 message queues.  
 240               **ELSE NO\_TEST\_SUPPORT**  
 241       **ELSE NO\_OPTION**

242 **GA\_mqPCTSOpenMaxFD**  
 243       **FOR:** *mq\_open*()  
 244       **M\_GA\_mqPCTSOpenMaxFD**()  
 245       Conformance for *mq\_hdr*: *PASS, NO\_OPTION*

## 246 15.2 Message Passing Functions

### 247 15.2.1 Open a Message Queue

248 Function: *mq\_open*()

#### 249 15.2.1.1 Synopsis

250 **1**  
 251       **M\_GA\_stdC\_proto\_decl**(*mqd\_t*; *mq\_open*; *const char \*name*, *int oflag*, ... *oflag*; *mqueue.h*;;;)  
 252       **SEE:** Assertion *GA\_stdC\_proto\_decl* in §2.7.3  
 253       Conformance for *mq\_open*: *PASS*[1, 2], *NO\_OPTION*

254 **2**  
 255       **M\_GA\_commonC\_int\_result\_decl**(*mqd\_t*; *mq\_open*; *mqueue.h*;;;)  
 256       **SEE:** Assertion *GA\_commonC\_int\_result\_decl* in §2.7.3

257 *Conformance for mq\_open: PASS[1, 2], NO\_OPTION*

258 **3**

259 *M\_GA\_macro\_result\_decl(mqd\_t; mq\_open; mqueue.h;;)*

260 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4

261 *Conformance for mq\_open: PASS, NO\_OPTION*

262 **4**

263 *M\_GA\_macro\_args (mq\_open; mqueue.h;;)*

264 **SEE:** Assertion GA\_macro\_args in §2.7.3

265 *Conformance for mq\_open: PASS, NO\_OPTION*

266 **15.2.1.2 Description**

267 *M\_GA\_mqOpenMaxFD()*

268 **SEE:** Assertion GA\_mqOpenMaxFD in §15.1.1.

269 *Conformance for mq\_open: PASS[OpenMaxMqs, PCTSopenMaxMqs]*

270 *M\_GA\_mqPCTSOpenMaxFD()*

271 **SEE:** Assertion GA\_mqPCTSOpenMaxFD in §15.1.1.

272 *Conformance for mq\_open: PASS[OpenMaxMqs, PCTSopenMaxMqs]*

273 **mq\_open**

274 **IF** PCTS\_mq\_open **THEN**

275 **IF** PCTS\_GAP\_mq\_open **THEN**

276 **TEST:** A successful call to mq\_open() establishes a connection between a process and

277 a message queue, name, and returns a message queue, descriptor which can be

278 used by other functions to refer to that message queue.

279 **ELSE** NO\_TEST\_SUPPORT

280 **ELSE** NO\_OPTION

281 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

282 **D\_1 IF** PCTS\_mq\_open and a PCD.1b documents the following **THEN**

283 **IF** PCTS\_GAP\_mq\_open **THEN**

284 **TEST:** A PCD.1b that documents whether the name appears in the file system and is

285 visible to other functions that take pathnames as arguments does so in §15.2.1.2.

286 **ELSE** NO\_TEST\_SUPPORT

287 **ELSE** NO\_OPTION

288 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

289 **5**

290 *M\_GA\_portableFilenames(mq\_open)*

291 **SEE:** Assertion GA\_portableFilenames in §2.2.2.40

292 *Conformance for mq\_open: PASS, NO\_OPTION*

293 **6**

294 *M\_GA\_upperLowerNames(mq\_open)*

295 **SEE:** Assertion GA\_upperLowerNames in §2.2.2.40

296 *Conformance for mq\_open: PASS, NO\_OPTION*

297 **7**

298 *M\_GA\_PRNoTrunc(mq\_open)*

299 **SEE:** Assertion GA\_PRNoTrunc in §2.3.6

300 *Conformance for mq\_open: PASS, NO\_OPTION*

301 **8**

302 *M\_GA\_PRNoTruncError(mq\_open)*

303 **SEE:** Assertion GA\_PRNoTruncError in §2.2.2.40

304 *Conformance for mq\_open: PASS, NO\_OPTION*

305 **9** **IF** *PCTS\_mq\_open* **THEN**

306 **IF** *PCTS\_GAP\_mq\_open* **THEN**

307 **TEST:** When *name* begins with the slash character, then processes calling

308 *mq\_open(name, oflag, ...)* with the same value of *name* refer to the same

309 message queue object, as long as that name has not been removed.

310 **TR:** Try the interface both in the same process that created the queue and in another

311 process.

312 **ELSE** *NO\_TEST\_SUPPORT*

313 **ELSE** *NO\_OPTION*

314 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

315 **D\_2** **IF** *PCTS\_mq\_open* **THEN**

316 **TEST:** The PCD.1b documents the effect of calling *mq\_open(name, oflag, ...)*, if *name* does

317 not begin with the slash character in §15.2.1.2.

318 **ELSE** *NO\_OPTION*

319 *Conformance for mq\_open: PASS, NO\_OPTION*

320 **D\_3** **IF** *PCTS\_mq\_open* **THEN**

321 **TEST:** The PCD.1b documents the interpretation of slash characters other than the leading

322 slash character in *name* in §15.2.1.2.

323 **ELSE** *NO\_OPTION*

324 *Conformance for mq\_open: PASS, NO\_OPTION*

325 **R\_1** **IF** *PCTS\_mq\_open* **THEN**

326 **TEST:** When the *name* argument is not the name of an existing message queue and *O\_CREAT*

327 is not set, *mq\_open(name, oflag, ...)* fails and returns an error.

328 **ELSE** *NO\_OPTION*

329 **SEE:** Assertion *mq\_open\_ENOENT* in §15.2.1.4

330 **10** **IF** *PCTS\_mq\_open* **THEN**

331 **IF** *PCTS\_GAP\_mq\_open* **THEN**

332 **TEST:** The access permission to receive messages or send messages as specified by *oflag*, is

333 granted if the calling process would be granted read or write access, respectively, to

334 an equivalently protected file.

335 Read and write access to files is determined as described in 2.3.

336 **ELSE** *NO\_TEST\_SUPPORT*

337 **ELSE** *NO\_OPTION*

338 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

339 **11** **FOR:** *mq\_open*

340 *M\_GA\_AP\_classAccess(mq\_open)*

341 **SEE:** Assertion *GA\_AP\_class Access* in §2.3.2

342 *Conformance for mq\_open: PASS*

343 **12** **FOR:** *mq\_open*

344 *M\_GA\_AP\_OVERRIDEFILEACCESS(mq\_open)*

345 **SEE:** Assertion *GA\_AP\_overrideFile Access* in §2.3.2

346 *Conformance for mq\_open: PASS*

347 **13** **IF** *PCTS\_mq\_open* **THEN**

348 **SETUP:** Include the header `<mqqueue.h>`.

349 **TEST:** The constants *O\_RDONLY*, *O\_WRONLY*, *O\_RDWR*, *O\_CREAT*, *O\_EXCL*, and *O\_NONBLOCK*

350 are defined and have different values.

351 **ELSE** *NO\_OPTION*

352 *Conformance for mq\_open: PASS, NO\_TEST*

353 **14** **IF** *PCTS\_mq\_open* **THEN**  
354 **IF** *PCTS\_mq\_send* and *PCTS\_mq\_receive* and *PCTS\_GAP\_mq\_open* **THEN**  
355 **TEST:** When a message queue is opened with the call *mq\_open*(*name*, *oflag*, ... ) and  
356 the flag *O\_RDONLY* is set in *oflag*, the process can use the returned message  
357 queue descriptor with *mq\_receive*(), but not with *mq\_send*().  
358 **ELSE** *NO\_TEST\_SUPPORT*  
359 **ELSE** *NO\_OPTION*  
360 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

361 **15** **IF** *PCTS\_mq\_open* **THEN**  
362 **IF** *PCTS\_GAP\_mq\_open* **THEN**  
363 **TEST:** A message queue may be opened multiple times in the same or different  
364 processes with the call *mq\_open*(*name*, *oflag*, ... ) and the flag *O\_RDONLY* set  
365 in *oflag*.  
366 **TR:** Try the interface both in the same process that created the queue and in another  
367 process.  
368 **ELSE** *NO\_TEST\_SUPPORT*  
369 **ELSE** *NO\_OPTION*  
370 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

371 **16** **IF** *PCTS\_mq\_open* **THEN**  
372 **IF** *PCTS\_mq\_open* and *PCTS\_mq\_send* and *PCTS\_mq\_receive* **THEN**  
373 **TEST:** When a message queue is opened with the call *mq\_open*(*name*, *oflag*, ... ) and  
374 the flag *O\_WRONLY* is set in *oflag*, the process can use the returned message  
375 queue descriptor with *mq\_send*() but not *mq\_receive*().  
376 **ELSE** *NO\_TEST\_SUPPORT*  
377 **ELSE** *NO\_OPTION*  
378 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

379 **17** **IF** *PCTS\_mq\_open* **THEN**  
380 **IF** *PCTS\_GAP\_mq\_open* **THEN**  
381 **TEST:** A message queue may be opened multiple times in the same or different  
382 processes with the call *mq\_open*(*name*, *oflag*, ... ) and the flag *O\_WRONLY* set  
383 in *oflag*.  
384 **ELSE** *NO\_TEST\_SUPPORT*  
385 **ELSE** *NO\_OPTION*  
386 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

387 **18** **IF** *PCTS\_mq\_open* **THEN**  
388 **IF** *PCTS\_mq\_open* and *PCTS\_mq\_send* and *PCTS\_mq\_receive* **THEN**  
389 **TEST:** When a message queue is opened with the call *mq\_open*(*name*, *oflag*, ... ) and  
390 the flag *O\_RDWR* is set in *oflag*, the process can use any of the functions allowed  
391 for *O\_RDONLY* and *O\_WRONLY*.  
392 **TR:** Try both *mq\_send*() and *mq\_receive*().  
393 **ELSE** *NO\_TEST\_SUPPORT*  
394 **ELSE** *NO\_OPTION*  
395 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

396 **19** **IF** *PCTS\_mq\_open* **THEN**  
397 **IF** *PCTS\_GAP\_mq\_open* **THEN**  
398 **TEST:** When a message queue is opened with the call *mq\_open*(*name*, *oflag*, ... ) and  
399 the flag *O\_RDWR* is set in *oflag*, it may be open multiple times in the same or  
400 different processes for sending messages.  
401 **ELSE** *NO\_TEST\_SUPPORT*  
402 **ELSE** *NO\_OPTION*  
403 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

404 **20** **IF** *PCTS\_mq\_open* **THEN**



405                   **IF** *PCTS\_GAP\_mq\_open* **THEN**  
406                    **TEST:**    When a message queue is opened with the call *mq\_open(name, oflag, mode, attr*  
407                                    ) the named message queue does not already exist, and the flag *O\_CREAT* is set  
408                                    in *oflag*, a message queue is created without any messages in it.  
409                    **ELSE NO\_TEST\_SUPPORT**  
410                   **ELSE NO\_OPTION**  
411                    Conformance for *mq\_open*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

412    **21**       **IF** *PCTS\_mq\_open* **THEN**  
413                    **IF** *PCTS\_GAP\_mq\_open* **THEN**  
414                                    **TEST:**    When *mq\_open(name, oflag, mode, attr )* is called, with *O\_CREAT* set and  
415                                    *O\_EXCL* cleared, and the named message queue already exists, the *O\_CREAT* flag  
416                                    has no effect.  
417                                    **ELSE NO\_TEST\_SUPPORT**  
418                    **ELSE NO\_OPTION**  
419                    Conformance for *mq\_open*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

420    **22**       **IF** *PCTS\_mq\_open* **THEN**  
421                    **IF** *PCTS\_GAP\_mq\_open* **THEN**  
422                                    **TEST:**    When a new message queue is created with *mq\_open()*, the user ID of the  
423                                    message queue is set to the effective user ID of the process.  
424                                    **ELSE NO\_TEST\_SUPPORT**  
425                    **ELSE NO\_OPTION**  
426                    Conformance for *mq\_open*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

427    **23**       **IF** *PCTS\_mq\_open* **THEN**  
428                    **IF** *PCTS\_GAP\_mq\_open* **THEN**  
429                                    **TEST:**    When a new message queue is created with *mq\_open()*, the group ID of the  
430                                    message queue is set to the effective group ID of the process.  
431                                    **ELSE NO\_TEST\_SUPPORT**  
432                    **ELSE NO\_OPTION**  
433                    Conformance for *mq\_open*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

434    **24**       **IF** *PCTS\_mq\_open* **THEN**  
435                    **IF** *PCTS\_GAP\_mq\_open* **THEN**  
436                                    **TEST:**    When a new message queue is created with *mq\_open(name, oflag, ... )*, the  
437                                    “file permission bits” are set to the value of *mode*.  
438                                    **TR:** Test for octal values 0000 through 0777.  
439                                    **ELSE NO\_TEST\_SUPPORT**  
440                    **ELSE NO\_OPTION**  
441                    Conformance for *mq\_open*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

442    **D\_4** **IF** *PCTS\_mq\_open* **THEN**  
443                    **TEST:**    The PCD.1b documents the effect when bits in *mode* other than file permission bits  
444                                    are set in §15.2.1.2.  
445                    **ELSE NO\_OPTION**  
446                    Conformance for *mq\_open*: *PASS, NO\_OPTION*

447    **D\_5** **IF** *PCTS\_mq\_open* **THEN**  
448                    **TEST:**    The PCD.1b documents the implementation-defined default message queue attributes  
449                                    with which message queues are created if *mq\_open(name, oflag, mode, attr )* is called  
450                                    and *attr* is *NULL*, in §15.2.1.2.  
451                    **ELSE NO\_OPTION**  
452                    Conformance for *mq\_open*: *PASS, NO\_OPTION*

453    **25**       **IF** *PCTS\_mq\_open* **THEN**  
454                    **IF** *PCTS\_GAP\_mq\_open* **THEN**  
455                                    **TEST:**    When *attr* is non-*NULL* and the calling process has the appropriate privilege on  
456                                    *name*, the message queue *mq\_maxmsg* and *mq\_msgsiz*e attributes are set to the

457 values of the corresponding members in the *mq\_attr* structure referred to by  
458 *attr*.  
459 **ELSE NO\_TEST\_SUPPORT**  
460 **ELSE NO\_OPTION**  
461 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

462 **R\_2 IF PCTS\_mq\_open THEN**  
463 **TEST:** When *attr* is non\_NULL, but the calling process does not have the appropriate  
464 privilege on *name*, the *mq\_open()* function fails and returns an error without creating  
465 the message queue.  
466 **ELSE NO\_OPTION**  
467 **SEE:** Assertion *mq\_open\_EACCESS2* in §15.2.1.4

468 **R\_3 IF PCTS\_mq\_open THEN**  
469 **TEST:** When O\_EXCL and O\_CREAT are set, *mq\_open()* fails if the message queue *name*  
470 exists.  
471 **ELSE NO\_OPTION**  
472 **SEE:** Assertion *mq\_open\_EEXIST* in §15.2.1.4

473 **26 IF PCTS\_mq\_open THEN**  
474 **IF PCTS\_GAP\_mq\_open THEN**  
475 **TEST:** The check for the existence of the message queue and the creation of the  
476 message queue if it does not exist is atomic with respect to other processes  
477 executing *mq\_open()* naming the same *name* with O\_EXCL and O\_CREAT set.  
478 There is no known reliable test method for this assertion.  
479 **ELSE NO\_TEST\_SUPPORT**  
480 **ELSE NO\_OPTION**  
481 *Conformance for mq\_open: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

482 **D\_6 IF PCTS\_mq\_open and a PCD.1b documents the following THEN**  
483 **TEST:** A PCD.1b that documents the result of calling *mq\_open(name, oflag, ...)* if O\_EXCL is  
484 set and O\_CREAT is not set does so in §15.2.2.2.  
485 **ELSE NO\_OPTION**  
486 *Conformance for mq\_open: PASS, NO\_OPTION*

487 **27 IF PCTS\_mq\_open THEN**  
488 **IF PCTS\_GAP\_mq\_open THEN**  
489 **TEST:** The setting of O\_NONBLOCK is associated with the open message queue  
490 descriptor and determines whether a *mq\_send()* or *mq\_receive()* waits for  
491 resources or messages that are not currently available, or fails with *errno* set to  
492 [EAGAIN].  
493 See *mq\_send()* and *mq\_receive()* for details.  
494 **ELSE NO\_TEST\_SUPPORT**  
495 **ELSE NO\_OPTION**  
496 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

497 **28 IF PCTS\_mq\_open THEN**  
498 **IF PCTS\_GAP\_mq\_open THEN**  
499 **TEST:** The *mq\_open()* function does not add or remove messages from the queue.  
500 **TR:** Open a message queue, send a single message, re-open the same queue, then try two  
501 successive reads.  
502 **ELSE NO\_TEST\_SUPPORT**  
503 **ELSE NO\_OPTION**  
504 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

505 **D\_7 IF PCTS\_mq\_open and a PCD.1b documents the following THEN**

506                   **TEST:**     A PCD.1b that documents whether or not it supports the *mq\_open()* function does so  
 507                                    in §15.2.1.2.  
 508                   **ELSE NO\_OPTION**  
 509                   *Conformance for mq\_open: PASS, NO\_OPTION*

### 510     **15.2.1.3 Returns**

511     **R\_4 IF PCTS\_mq\_open THEN**  
 512                   **TEST:**     When a call to *mq\_open()* completes successfully, the function returns a message  
 513                                    queue descriptor.  
 514                   **ELSE NO\_OPTION**  
 515                   **SEE:**       Assertion *mq\_open* in §15.2.1.2

516     **R\_5 IF PCTS\_mq\_open THEN**  
 517                   **TEST:**     When a call to *mq\_open()* completes unsuccessfully, the function returns (*mqd\_t*) -1  
 518                                    and sets *errno* to indicate the error.  
 519                   **ELSE NO\_OPTION**  
 520                   **SEE:**       All assertions in §15.2.1.4

### 521     **15.2.1.4 Errors**

522     **mq\_open\_EACCESS1**  
 523                   **IF PCTS\_mq\_open THEN**  
 524                    **IF PCTS\_GAP\_mq\_open THEN**  
 525                      **TEST:**     A call to *mq\_open(name, oflag, ...)*, when the message queue exists and the  
 526                                    permissions specified by *oflag* are denied, returns a value of -1 and sets *errno*  
 527                                    to [EACCESS].  
 528                    **ELSE NO\_TEST\_SUPPORT**  
 529                   **ELSE NO\_OPTION**  
 530                   *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

531     **mq\_open\_EACCESS2**  
 532                   **IF PCTS\_mq\_open THEN**  
 533                    **IF PCTS\_RAP\_mq\_open THEN**  
 534                      **TEST:**     A call to *mq\_open(name, oflag, ...)*, when the message queue does not exist and  
 535                                    permission to create the message queue is denied, returns a value of -1 and sets  
 536                                    *errno* to [EACCESS].  
 537                    **ELSE NO\_TEST\_SUPPORT**  
 538                   **ELSE NO\_OPTION**  
 539                   *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

540     **mq\_open\_EEXIT**  
 541                   **IF PCTS\_mq\_open THEN**  
 542                    **IF PCTS\_GAP\_mq\_open THEN**  
 543                      **TEST:**     A call to *mq\_open(name, oflag, ...)*, when O\_CREAT and O\_EXCL are set and the  
 544                                    named message queue already exists, returns a value of -1 and sets *errno* to  
 545                                    [EEXIST].  
 546                    **ELSE NO\_TEST\_SUPPORT**  
 547                   **ELSE NO\_OPTION**  
 548                   *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

549     **29 IF PCTS\_mq\_open THEN**  
 550                    **IF PCTS\_GAP\_mq\_open THEN**  
 551                      **TEST:**     A call to *mq\_open()* when the *mq\_open()* operation is interrupted by a signal,  
 552                                    returns a value of -1 and sets *errno* to [EINTR].  
 553                    **ELSE NO\_TEST\_SUPPORT**  
 554                   **ELSE NO\_OPTION**

555 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

556 **30** **IF** *PCTS\_mq\_open* **THEN**  
557 **IF** *PCTS\_GAP\_mq\_open* **THEN**  
558 **TEST:** A call to *mq\_open()*, when the *mq\_open(name, oflag, ...)* operation is not  
559 supported for the given name, returns a value of -1 and sets *errno* to [EINVAL].  
560 **ELSE** *NO\_TEST\_SUPPORT*  
561 **ELSE** *NO\_OPTION*  
562 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

563 **D-8** **IF** *PCTS\_mq\_open* **THEN**  
564 **TEST:** The PCD.1b documents under what circumstances this error may be returned in  
565 §15.2.1.4  
566 **ELSE** *NO\_OPTION*  
567 *Conformance for mq\_open: PASS, NO\_OPTION*

568 **31** **IF** *PCTS\_mq\_open* **THEN**  
569 **IF** *PCTS\_GAP\_mq\_open* **THEN**  
570 **TEST:** A call to *mq\_open(name, oflag, ...)*, when *O\_CREAT* is specified in *oflag*, the  
571 value of *attr* is not *NULL*, and *mq\_maxmsg* is less than or equal to zero, returns  
572 a value of -1 and sets *errno* to [EINVAL].  
573 **ELSE** *NO\_TEST\_SUPPORT*  
574 **ELSE** *NO\_OPTION*  
575 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

576 **32** **IF** *PCTS\_mq\_open* **THEN**  
577 **IF** *PCTS\_GAP\_mq\_open* **THEN**  
578 **TEST:** A call to *mq\_open(name, oflag, ...)*, when *O\_CREAT* is specified in *oflag*, the  
579 value of *attr* is not *NULL*, and *mq\_msgsize* is less than or equal to zero, returns  
580 a value of -1 and sets *errno* to [EINVAL].  
581 **ELSE** *NO\_TEST\_SUPPORT*  
582 **ELSE** *NO\_OPTION*  
583 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

584 **33** **IF** *PCTS\_mq\_open* **THEN**  
585 **IF** *PCTS\_GAP\_mq\_open* and ( {OPEN\_MAX} ≤ *PCTS\_OPEN\_MAX* ) **THEN**  
586 **TEST:** A call to *mq\_open()*, when too many message queue descriptors or file  
587 descriptors are currently in use by this process, returns a value of -1 and sets  
588 *errno* to [EMFILE].  
589 **ELSE** *NO\_TEST\_SUPPORT*  
590 **ELSE** *NO\_OPTION*  
591 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

592 **34** **IF** *PCTS\_mq\_open* **THEN**  
593 **IF** *PCTS\_GAP\_mq\_open* and {MQ\_OPEN\_MAX} ≤ *PCTS\_MQ\_OPEN\_MAX* **THEN**  
594 **TEST:** A call to *mq\_open()*, when too many message queue descriptors or file  
595 descriptors are currently in use by this process, returns a value of -1 and sets  
596 *errno* to [EMFILE].  
597 **ELSE** *NO\_TEST\_SUPPORT*  
598 **ELSE** *NO\_OPTION*  
599 *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

600 **35** **IF** *PCTS\_mq\_open* **THEN**  
601 **IF** *PCTS\_GAP\_mq\_open* and { \_POSIX\_NO\_TRUNC } and ( {PATH\_MAX} ≤ *PCTS\_PATH\_MAX* )  
602 **THEN**  
603 **TEST:** A call to *mq\_open(name, oflag, ...)*, when the length of the *name* string exceeds  
604 {PATH\_MAX}, while { \_POSIX\_NO\_TRUNC } is in effect, returns a value of -1 and  
605 sets *errno* to [ENAMETOOLONG].

606                   **ELSE NO\_TEST\_SUPPORT**  
607                   **ELSE NO\_OPTION**  
608                   *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

609   **36**           **IF PCTS\_mq\_open THEN**  
610                   **IF PCTS\_GAP\_mq\_open and {\_POSIX\_NO\_TRUNC} and ( {NAME\_MAX} <= PCTS\_NAME\_MAX)**  
611                   **THEN**  
612                   **TEST:**    A call to *mq\_open* (*name, oflag, ...*), when a pathname component is longer  
613                                   than {NAME\_MAX}, while {\_POSIX\_NO\_TRUNC} is in effect, returns a value of  
614                                   -1 and sets *errno* to [ENAMETOOLONG].  
615                   **ELSE NO\_TEST\_SUPPORT**  
616                   **ELSE NO\_OPTION**  
617                   *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

618   **37**           **IF PCTS\_mq\_open THEN**  
619                   **IF PCTS\_GAP\_mq\_open THEN**  
620                   **TEST:**    A call to *mq\_open* (), when too many message queues are currently open in the  
621                                   system, returns a value of -1 and sets *errno* to [ENFILE].  
622                   **ELSE NO\_TEST\_SUPPORT**  
623                   **ELSE NO\_OPTION**  
624                   *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

625   **mq\_open\_ENOENT**  
626                   **IF PCTS\_mq\_open THEN**  
627                   **IF PCTS\_GAP\_mq\_open THEN**  
628                   **TEST:**    A call to *mq\_open* (), when O\_CREAT is not set and the named message queue  
629                                   does not exist, returns a value of -1 and sets *errno* to [ENOENT].  
630                   **ELSE NO\_TEST\_SUPPORT**  
631                   **ELSE NO\_OPTION**  
632                   *Conformance for mq\_open: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

633   **38**           **IF PCTS\_mq\_open THEN**  
634                   **IF PCTS\_GAP\_mq\_open THEN**  
635                   **TEST:**    A call to *mq\_open* (), when there is insufficient space for the creation of the new  
636                                   message queue, returns a value of -1 and sets *errno* to [ENOSPC].  
637                                   There is no known reliable test method for this assertion.  
638                   **ELSE NO\_TEST\_SUPPORT**  
639                   **ELSE NO\_OPTION**  
640                   *Conformance for mq\_open: PASS, NO\_TEST, NO\_TEST\_SUPPORT, NO\_OPTION*

641   **39**           **IF not PCTS\_mq\_open THEN**  
642                   **TEST:**    A call to *mq\_open* () returns a value of -1 and sets *errno* to [ENOSYS].  
643                   **ELSE NO\_TEST\_SUPPORT**  
644                   **ELSE NO\_OPTION**  
645                   *Conformance for mq\_open: PASS, NO\_OPTION*

## 646   **15.2.2**    **Close a Message Queue**

647   Function: *mq\_close*()

### 648   **15.2.2.1** **Synopsis**

649   **1**  
650                   *M\_GA\_stdC\_proto\_decl(int; mq\_close; mqd\_t mqdes; mqueue.h;;)*  
651                   **SEE:**    Assertion GA\_stdC\_proto\_decl in §2.7.3  
652                   *Conformance for mq\_close: PASS[1, 2], NO\_OPTION*

653     **2**  
654     *M\_GA\_commonC\_int\_result\_decl(mq\_close; mqueue.h;;)*  
655     **SEE:**     Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
656     Conformance for *mq\_close*: *PASS[1, 2], NO\_OPTION*

657     **3**  
658     *M\_GA\_macro\_result\_decl(int; mq\_close; mqueue.h;;)*  
659     **SEE:**     Assertion GA\_macro\_result\_decl in §1.3.4  
660     Conformance for *mq\_close*: *PASS, NO\_OPTION*

661     **4**  
662     *M\_GA\_macro\_args ( mq\_close; mqueue.h;;)*  
663     **SEE:**     Assertion GA\_macro\_args in §2.7.3  
664     Conformance for *mq\_close*: *PASS, NO\_OPTION*

665     **15.2.2.2 Description**

666     **mq\_close**

667     **IF** *PCTS\_mq\_close* **THEN**  
668         **TEST:**     A call to *mq\_close()* removes the association between the message queue descriptor,  
669                     *mqdes*, and its message queue, and returns a value of zero.  
670     **ELSE** *NO\_OPTION*  
671     Conformance for *mq\_close*: *PASS, NO\_OPTION*

672     **D\_1** **IF** *PCTS\_mq\_close* and a PCD.1b documents the following **THEN**  
673         **TEST:**     A PCD.1b that documents the results of using this message queue descriptor after  
674                     successful return from this *mq\_close()* and until the return of this message queue  
675                     descriptor from a subsequent *mq\_open()* does so in §15.2.2.2.  
676     **ELSE** *NO\_OPTION*  
677     Conformance for *mq\_close*: *PASS, NO\_OPTION*

678     **5**     **IF** *PCTS\_mq\_close* **THEN**  
679         **IF** *PCTS\_mq\_notify* **THEN**  
680             **TEST:**     When the process has successfully attached a notification request to the message  
681                     queue via *mqdes*, a call to *mq\_close()* removes this attachment and makes the  
682                     message queue available for another process to attach for notification.  
683         **ELSE** *NO\_TEST\_SUPPORT*  
684     **ELSE** *NO\_OPTION*  
685     Conformance for *mq\_close*: *PASS,NO\_TEST\_SUPPORT, NO\_OPTION*

686     **D\_2** **IF** *PCTS\_mq\_close* and a PCD.1b documents the following **THEN**  
687         **TEST:**     A PCD.1b that documents whether or not it supports the *mq\_close()* function does so  
688                     in §15.2.2.2.  
689     **ELSE** *NO\_OPTION*  
690     Conformance for *mq\_close*: *PASS, NO\_OPTION*

691     **15.2.2.3 Returns**

692     **R\_1** **IF** *PCTS\_mq\_close* **THEN**  
693         **TEST:**     When a call to *mq\_close()* completes successfully, it returns a value of 0.  
694     **ELSE** *NO\_OPTION*  
695     **SEE:**     Assertion *mq\_close* in §15.2.2.2

696     **R\_2** **IF** *PCTS\_mq\_close* **THEN**  
697         **TEST:**     When a call to *mq\_close()* completes unsuccessfully, the function returns a value of  
698                     -1 and sets *errno* to indicate the error.

699           **ELSE NO\_OPTION**  
700           **SEE:**     All assertions in §15.2.2.4

701   **15.2.2.4 Errors**

702   **6**       **IF PCTS\_mq\_close THEN**  
703           **TEST:**    A call to *mq\_close(mqdes)*, when the *mqdes* argument is not a valid message queue  
704                        descriptor, returns a value of -1 and sets *errno* to [EBADF].  
705           **ELSE NO\_OPTION**  
706           *Conformance for mq\_close: PASS, NO\_OPTION*

707   **7**       **IF not PCTS\_mq\_close THEN**  
708           **TEST:**    A call to *mq\_close()*, returns a value of -1 and sets *errno* to [ENOSYS].  
709           **ELSE NO\_OPTION**  
710           *Conformance for mq\_close: PASS, NO\_OPTION*

711   **15.2.3    Remove a Message Queue**

712   Function: *mq\_unlink()*

713   **15.2.3.1 Synopsis**

714   **1**  
715        *M\_GA\_stdC\_proto\_decl(int; mq\_unlink; const char \*name; mqueue.h;;)*  
716        **SEE:**     Assertion GA\_stdC\_proto\_decl in §2.7.3  
717        *Conformance for mq\_unlink: PASS[1, 2], NO\_OPTION*

718   **2**  
719        *M\_GA\_commonC\_int\_result\_decl(mq\_unlink; mqueue.h;;)*  
720        **SEE:**     Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
721        *Conformance for mq\_unlink: PASS[1, 2], NO\_OPTION*

722   **3**  
723        *M\_GA\_macro\_result\_decl(int; mq\_unlink; mqueue.h;;)*  
724        **SEE:**     Assertion GA\_macro\_result\_decl in §1.3.4  
725        *Conformance for mq\_unlink: PASS, NO\_OPTION*

726   **4**  
727        *M\_GA\_macro\_args ( mq\_unlink; mqueue.h;;)*  
728        **SEE:**     Assertion GA\_macro\_args in §2.7.3  
729        *Conformance for mq\_unlink: PASS, NO\_OPTION*

730   **15.2.3.2 Description**

731   **mq\_unlink**  
732        **IF PCTS\_mq\_unlink THEN**  
733           **TEST:**    A successful call to *mq\_link()* removes the message queue named by the pathname  
734                        *name*, and returns a value of zero.  
735        **ELSE NO\_OPTION**  
736        *Conformance for mq\_unlink: PASS, NO\_OPTION*

737   **R\_1 IF PCTS\_mq\_unlink THEN**  
738           **IF PCTS\_mq\_open and PCTS\_gap\_mq\_open THEN**  
739           **TEST:**    After a successful call to *mq\_unlink(name)* a call to *mq\_open(name, oflag, ...)* fails  
740                        if the flag O\_CREAT is not set in *flags*.  
741           **ELSE NO\_TEST\_SUPPORT**

742           **ELSE NO\_OPTION**  
743           **SEE:**       Assertion mq\_unlink\_ENOENT in §15.2.3.4

744   **5**       **IF PCTS\_mq\_unlink THEN**  
745           **TEST:**     When one or more processes have the message queue open when *mq\_unlink()* is  
746                       called, destruction of the message queue is postponed until all references to the  
747                       message queue have been closed.  
748           **ELSE NO\_OPTION**  
749           Conformance for *mq\_unlink*: *PASS, NO\_OPTION*

750   **D\_1 IF PCTS\_mq\_unlink** and a PCD.1b documents the following **THEN**  
751           **TEST:**     A PCD.1b that documents whether or not it supports the *mq\_link()* function does so  
752                       in §15.2.3.4.  
753           **ELSE NO\_OPTION**  
754           Conformance for *mq\_unlink*: *PASS, NO\_OPTION*

755   **15.2.3.3 Returns**

756   **R\_2 IF PCTS\_mq\_unlink THEN**  
757           **TEST:**     When a call to *mq\_link()* completes successfully, the function returns a value of 0.  
758           **ELSE NO\_OPTION**  
759           **SEE:**       Assertion mq\_unlink in §15.2.3.2

760   **R\_3 IF PCTS\_mq\_unlink THEN**  
761           **TEST:**     When a call to *mq\_link()* completes unsuccessfully, the named message queue is  
762                       unchanged and the function returns a value of -1 and sets *errno* to indicate the error.  
763           **ELSE NO\_OPTION**  
764           **SEE:**       All assertions in §15.2.3.4

765   **15.2.3.4 Errors**

766   **6**       **IF PCTS\_mq\_unlink THEN**  
767           **TEST:**     A call to *mq\_unlink(name)*, when permission is denied to unlink the named message  
768                       queue, returns a value of -1 and sets *errno* to [EACCES].  
769           **ELSE NO\_OPTION**  
770           Conformance for *mq\_unlink*: *PASS, NO\_OPTION*

771   **7**       **IF PCTS\_mq\_unlink THEN**  
772           **IF { \_POSIX\_NO\_TRUNC } and ( { NAME\_MAX } <= PCTS\_NAME\_MAX ) THEN**  
773           **TEST:**     A call to *mq\_unlink(name)*, when the length of the *name* string exceeds {NAME\_MAX}  
774                       while { \_POSIX\_NO\_TRUNC } is in effect, returns a value of -1 and sets *errno* to  
775                       [ENAMETOOLONG].  
776           **ELSE NO\_TEST\_SUPPORT**  
777           **ELSE NO\_OPTION**  
778           Conformance for *mq\_unlink*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

779   **mq\_unlink\_ENOENT**  
780           **IF PCTS\_mq\_unlink THEN**  
781           **TEST:**     A call to *mq\_unlink(name)*, when the named message queue does not exist, returns a  
782                       value of -1 and sets *errno* to [ENOENT].  
783           **ELSE NO\_OPTION**  
784           Conformance for *mq\_unlink*: *PASS, NO\_OPTION*

785   **8**       **IF not PCTS\_mq\_unlink THEN**  
786           **TEST:**     A call to *mq\_unlink()* returns a value of -1 and sets *errno* to [ENOSYS].  
787           **ELSE NO\_OPTION**



788 *Conformance for mq\_unlink: PASS, NO\_OPTION*

## 789 15.2.4 Send a Message to a Message Queue

790 Function: *mq\_send()*

### 791 15.2.4.1 Synopsis

792 **1**

793 *M\_GA\_stdC\_proto\_decl(int; mq\_send; mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len, unsigned*  
794 *int msg\_prio; mqueue.h;;)*

795 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3

796 *Conformance for mq\_send: PASS[1, 2], NO\_OPTION*

797 **2**

798 *M\_GA\_commonC\_int\_result\_decl(mq\_send; mqueue.h;;)*

799 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3

800 *Conformance for mq\_send: PASS[1, 2], NO\_OPTION*

801 **3**

802 *M\_GA\_macro\_result\_decl(int; mq\_send; mqueue.h;;)*

803 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4

804 *Conformance for mq\_send: PASS, NO\_OPTION*

805 **4**

806 *M\_GA\_macro\_args ( mq\_send; mqueue.h;;)*

807 **SEE:** Assertion GA\_macro\_args in §2.7.3

808 *Conformance for mq\_send: PASS, NO\_OPTION*

### 809 15.2.4.2 Description

#### 810 **mq\_send**

811 **IF PCTS\_mq\_send THEN**

812 **TEST:** A successful call to *mq\_send()* adds the message pointed to by the argument *msg\_ptr*  
813 to the message queue specified by *mqdes*, and returns a value of zero.

814 **ELSE NO\_OPTION**

815 *Conformance for mq\_send: PASS, NO\_OPTION*

816 **R\_1 IF PCTS\_mq\_send THEN**

817 **TEST:** When the value of *msg\_len*, which specifies the length of the message in bytes pointed  
818 to by *msg\_ptr*, is greater than the *mq\_msgsiz*e attribute of the message queue,  
819 *mq\_send()* fails.

820 **ELSE NO\_OPTION**

821 **SEE:** Assertion *mq\_send\_EMMSGSIZE* in §15.2.4.4

822 **5 IF PCTS\_mq\_send THEN**

823 **TEST:** When the specified message queue is not full, *mq\_send()* behaves as if the message  
824 is inserted into the message queue at the position indicated by the *msg\_prio* argument.

825 **ELSE NO\_OPTION**

826 *Conformance for mq\_send: PASS, NO\_OPTION*

827 **6 IF PCTS\_mq\_send THEN**

828 **TEST:** A message with a larger numeric value of *msg\_prio* is inserted before messages with  
829 lower values of *msg\_prio*.

830 **ELSE NO\_OPTION**

831 *Conformance for mq\_send: PASS, NO\_OPTION*

832 **7 IF PCTS\_mq\_send THEN**  
 833 **TEST:** A message is inserted after other messages in the queue, if any, with equal *msg\_prio*.  
 834 **ELSE NO\_OPTION**  
 835 *Conformance for mq\_send: PASS, NO\_OPTION*

836 **R\_2 IF PCTS\_mq\_send THEN**  
 837 **TEST:** The value of *msg\_prio* is less than or equal to {MQ\_PRIO\_MAX}.  
 838 **ELSE NO\_OPTION**  
 839 **SEE:** Assertion *mq\_send\_EINVAL* in §15.2.4.4

840 **8 IF PCTS\_mq\_send THEN**  
 841 **TEST:** When the specified message queue is full and O\_NONBLOCK is not set in the message  
 842 queue description associated with *mqdes*, *mq\_send()* blocks until space becomes  
 843 available to enqueue the message, or until *mq\_send()* is interrupted by a signal.  
 844 **ELSE NO\_OPTION**  
 845 *Conformance for mq\_send: PASS, NO\_OPTION*

846 **9 IF PCTS\_mq\_send THEN**  
 847 **IF { \_POSIX\_PRIORITY\_SCHEDULING } THEN**  
 848 **TEST:** When more than one process is waiting to send when space becomes available in the  
 849 message queue and the { \_POSIX\_PRIORITY\_SCHEDULING } option is supported, then the  
 850 process of the highest priority that has been waiting the longest unblocked to send its  
 851 message.  
 852 **ELSE NO\_TEST\_SUPPORT**  
 853 **ELSE NO\_OPTION**  
 854 *Conformance for mq\_send: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

855 **D\_1 IF PCTS\_mq\_send and a PCD.1b documents the following THEN**  
 856 **TEST:** A PCD.1b that documents which waiting process is unblocked otherwise does so in  
 857 §15.2.4.2.  
 858 **ELSE NO\_OPTION**  
 859 *Conformance for mq\_send: PASS, NO\_OPTION*

860 **R\_3 IF PCTS\_mq\_send THEN**  
 861 **TEST:** When the specified message queue is full and O\_NONBLOCK is set in the message  
 862 queue description associated with *mqdes*, the message is not queued and *mq\_send()*  
 863 returns an error.  
 864 **ELSE NO\_OPTION**  
 865 **SEE:** Assertion *mq\_send\_EAGAIN* in §15.2.4.4

866 **D\_2 IF PCTS\_mq\_send and a PCD.1b documents the following THEN**  
 867 **TEST:** A PCD.1b that documents whether or not it supports the *mq\_send()* function does so  
 868 in §15.2.4.2.  
 869 **ELSE NO\_OPTION**  
 870 *Conformance for mq\_send: PASS, NO\_OPTION*

### 871 **15.2.4.3 Returns**

872 **R\_4 IF PCTS\_mq\_send THEN**  
 873 **TEST:** When a call to *mq\_send()* completes successfully, it returns a value of 0.  
 874 **ELSE NO\_OPTION**  
 875 **SEE:** Assertion *mq\_send* in §15.2.4.2

876 **R\_5 IF PCTS\_mq\_send THEN**  
 877 **TEST:** When a call to *mq\_send()* completes unsuccessfully, no message is enqueued, the  
 878 function returns -1, and *errno* is set to indicate the error.

879           **ELSE NO\_OPTION**  
880           **SEE:**     All assertions in §15.2.4.4

881   **15.2.4.4   Errors**

882   **mq\_send\_EAGAIN**  
883       **IF PCTS\_mq\_send THEN**  
884           **TEST:**    A call to *mq\_send(mqdes, msg\_ptr, msg\_len, msg\_prio)*, when the *O\_NONBLOCK* flag  
885                      is set in the message queue description associated with *mqdes*, and the specified  
886                      message queue is full, returns a value of -1 and sets *errno* to [EAGAIN].  
887       **ELSE NO\_OPTION**  
888       Conformance for *mq\_send*: *PASS, NO\_OPTION*

889   **10       IF PCTS\_mq\_send THEN**  
890           **TEST:**    A call to *mq\_send(mqdes, msg\_ptr, msg\_len, msg\_prio)*, when the *mqdes* argument  
891                      is not a valid message queue descriptor open for writing, returns a value of -1 and sets  
892                      *errno* to [EBADF].  
893           **TR:**    Open a message queue *O\_RDONLY*, then send to that queue.

894                                      Send to an invalid message queue.  
895       **ELSE NO\_OPTION**  
896       Conformance for *mq\_send*: *PASS, NO\_OPTION*

897   **11       IF PCTS\_mq\_send THEN**  
898           **TEST:**    A call to *mq\_send()*, when a signal interrupts the call, returns a value of -1 and sets  
899                      *errno* to [EINTR].  
900       **ELSE NO\_OPTION**  
901       Conformance for *mq\_send*: *PASS, NO\_OPTION*

902   **mq\_send\_EINVAL**  
903       **IF PCTS\_mq\_send THEN**  
904           **IF {MQ\_PRIO\_MAX} < {UINT\_MAX} THEN**  
905           **TEST:**    A call to *mq\_send(mqdes, msg\_ptr, msg\_len, msg\_prio)*, when the value of *msg\_prio*  
906                      is outside the valid range, returns a value of -1 and sets *errno* to [EINVAL].  
907           **TR:**    Try a value of {MQ\_PRIO\_MAX}+1, if it is less than {UINT\_MAX}  
908           **ELSE NO\_TEST\_SUPPORT**  
909       **ELSE NO\_OPTION**  
910       Conformance for *mq\_send*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

911   **mq\_send\_MSGSIZE**  
912       **IF PCTS\_mq\_send THEN**  
913           **TEST:**    A call to *mq\_send(mqdes, msg\_ptr, msg\_len, msg\_prio)*, when the specified message  
914                      length, *msg\_len*, exceeds the message size attribute of the message queue, returns a  
915                      value of -1 and sets *errno* to [EMSGSIZE].  
916           **ELSE NO\_TEST\_SUPPORT**  
917       **ELSE NO\_OPTION**  
918       Conformance for *mq\_send*: *PASS, NO\_OPTION*

919   **12       IF not PCTS\_mq\_send THEN**  
920           **TEST:**    A call to *mq\_send()* returns a value of -1 and sets *errno* to [ENOSYS].  
921       **ELSE NO\_OPTION**  
922       Conformance for *mq\_send*: *PASS, NO\_OPTION*

923   **15.2.5    Receive a Message From a Message Queue**

924   Function: *mq\_receive()*

925 **15.2.5.1 Synopsis**926 **1**927 *M\_GA\_stdC\_proto\_decl(ssize\_t; mq\_receive; mqd\_t mqdes, char \*msg\_ptr, size\_t msg\_len, unsigned*  
928 *int \*msg\_prio; mqueue.h;;)*929 **SEE:** Assertion GA\_stdC\_proto\_decl in §2.7.3930 *Conformance for mq\_receive: PASS[1, 2], NO\_OPTION*931 **2**932 *M\_GA\_commonC\_int\_result\_decl(ssize\_t; mq\_receive; mqueue.h;;)*933 **SEE:** Assertion GA\_commonC\_int\_result\_decl in §2.7.3934 *Conformance for mq\_receive: PASS[1, 2], NO\_OPTION*935 **3**936 *M\_GA\_macro\_result\_decl(ssize\_t; mq\_receive; mqueue.h;;)*937 **SEE:** Assertion GA\_macro\_result\_decl in §1.3.4938 *Conformance for mq\_receive: PASS, NO\_OPTION*939 **4**940 *M\_GA\_macro\_args (mq\_receive; mqueue.h;;)*941 **SEE:** Assertion GA\_macro\_args in §2.7.3942 *Conformance for mq\_receive: PASS, NO\_OPTION*943 **15.2.5.2 Description**944 **mq\_receive**945 **IF** PCTS\_mq\_receive **THEN**946 **TEST:** A successful call to *mq\_receive* () receives the oldest of the highest priority  
947 message(s) from the message queue specified by *mqdes*, copying it to the buffer  
948 pointed to by the *msg\_ptr* argument, removing it from the queue, and returning its  
949 length in bytes.950 **ELSE** NO\_OPTION951 *Conformance for mq\_receive: PASS, NO\_OPTION*952 **5**953 **IF** PCTS\_mq\_receive **THEN**954 **TEST:** When the size of the buffer in bytes, specified by the *msg\_len* argument, is less than  
955 the *mq\_msgsize* attribute of the message queue, the function *mq\_receive(mqdes,*  
*msg\_ptr, msg\_len, msg\_prio)* fails and returns an error.956 **ELSE** NO\_OPTION957 *Conformance for mq\_receive: PASS, NO\_OPTION*958 **6**959 **IF** PCTS\_mq\_receive **THEN**960 **TEST:** When the argument *msg\_prio* is not NULL, the priority of the selected message is  
961 stored in the location referenced by *msg\_prio*.962 **ELSE** NO\_OPTION963 *Conformance for mq\_receive: PASS, NO\_OPTION*964 **7**965 **IF** PCTS\_mq\_receive **THEN**966 **TEST:** When the specified message queue is empty and O\_NONBLOCK is not set in the  
967 message queue description associated with *mqdes*, *mq\_receive()* blocks until a  
968 message is enqueued on the message queue or until *mq\_receive()* is interrupted by a  
969 signal.970 **ELSE** NO\_OPTION971 *Conformance for mq\_receive: PASS, NO\_OPTION*972 **8**973 **IF** PCTS\_mq\_receive **THEN**974 **IF** { \_POSIX\_PRIORITY\_SCHEDULING } **THEN**

972                   **TEST:**     When more than one process is waiting to receive a message when a message arrives  
 973                                   at an empty queue and the `{_POSIX_PRIORITY_SCHEDULING}` option is supported, then  
 974                                   the process of highest priority that has been waiting the longest is selected to receive  
 975                                   the message.

976                   **ELSE NO\_TEST\_SUPPORT**

977                   **ELSE NO\_OPTION**

978                   *Conformance for mq\_receive: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

979   **D\_1 IF PCTS\_mq\_receive** and a PCD.1b documents the following **THEN**

980                   **TEST:**     A PCD.1b that documents which waiting process receives the message if more than one  
 981                                   process is waiting to receive a message when a message arrives at an empty queue and  
 982                                   the `{_POSIX_PRIORITY_SCHEDULING}` option is not supported, does so in §15.2.5.2.

983                   **ELSE NO\_OPTION**

984                   *Conformance for mq\_receive: PASS, NO\_OPTION*

985   **R\_1 IF PCTS\_mq\_receive THEN**

986                   **TEST:**     When the specified message queue is empty and `O_NONBLOCK` is set in the message  
 987                                   queue description associated with *mqdes*, no message is removed from the queue, and  
 988                                   *mq\_receive()* returns an error.

989                   **ELSE NO\_OPTION**

990                   **SEE:** Assertion for *mq\_receive\_EAGAIN* in §15.2.5.4

991   **D\_2 IF PCTS\_mq\_receive** and a PCD.1b documents the following **THEN**

992                   **TEST:**     A PCD.1b that documents whether or not it supports the *mq\_receive()* function does  
 993                                   so in §15.2.5.2.

994                   **ELSE NO\_OPTION**

995                   *Conformance for mq\_receive: PASS, NO\_OPTION*

### 996   **15.2.5.3 Returns**

997   **R\_2 IF PCTS\_mq\_receive THEN**

998                   **TEST:**     When a call to *mq\_receive()* completes successfully, it returns the length of the  
 999                                   selected message in bytes and the message is removed from the queue.

1000                   **ELSE NO\_OPTION**

1001                   **SEE:** Assertion for *mq\_receive* in §15.2.5.2

1002   **R\_3 IF PCTS\_mq\_receive THEN**

1003                   **TEST:**     When a call to *mq\_receive()* completes unsuccessfully, no message is removed from  
 1004                                   the queue, the function returns a value of -1, and sets *errno* to indicate the error.

1005                   **ELSE NO\_OPTION**

1006                   **SEE:**     All assertions in §15.2.5.4

### 1007   **15.2.5.4 Errors**

#### 1008   **mq\_receive\_EAGAIN**

1009                   **IF PCTS\_mq\_receive THEN**

1010                   **TEST:**     A test to *mq\_receive(mqdes, msg\_ptr, msg\_len, msg\_prio)*, when `O_NONBLOCK` is set  
 1011                                   in the message description associated with *mqdes*, and the specified message queue  
 1012                                   is empty, returns a value of -1 and sets *errno* to `[EAGAIN]`.

1013                   **ELSE NO\_OPTION**

1014                   *Conformance for mq\_receive: PASS, NO\_OPTION*

1015   **9 IF PCTS\_mq\_receive THEN**

1016                   **TEST:**     A call to *mq\_receive(mqdes, msg\_ptr, msg\_len, msg\_prio)*, when the *mqdes* argument  
 1017                                   is not a valid message queue descriptor open for reading, returns a value of -1 and sets  
 1018                                   *errno* to `[EBADF]`.

1019                   **TR:**     Open a message queue `O_WRONLY`, then receive from that queue.

1020                                   Receive from an invalid message queue.  
1021       **ELSE NO\_OPTION**  
1022       Conformance for *mq\_receive*: *PASS, NO\_OPTION*

1023       **10**       **IF PCTS\_mq\_receive THEN**  
1024               **TEST:**    A call to *mq\_receive(mqdes, msg\_ptr, msg\_len, msg\_prio)*, when the specified  
1025                           message buffer size, *msg\_len*, is less than the message size attribute of the message  
1026                           queue, returns a value of -1 and sets *errno* to [EMSGSIZE].  
1027       **ELSE NO\_OPTION**  
1028       Conformance for *mq\_receive*: *PASS, NO\_OPTION*

1029       **11**       **IF PCTS\_mq\_receive THEN**  
1030               **TEST:**    A call to *mq\_receive()*, when operation is interrupted by a signal, returns a value of  
1031                           -1 and sets *errno* to [EINTR].  
1032       **ELSE NO\_OPTION**  
1033       Conformance for *mq\_receive*: *PASS, NO\_OPTION*

1034       **12**       **IF not PCTS\_mq\_receive THEN**  
1035               **TEST:**    A call to *mq\_receive()* returns a value of -1 and sets *errno* to [ENOSYS].  
1036       **ELSE NO\_OPTION**  
1037       Conformance for *mq\_receive*: *PASS, NO\_OPTION*

1038       **13**       **IF PCTS\_mq\_receive THEN**  
1039               **IF PCTS\_DETECT\_MESSAGE\_DATA\_CORRUPTION THEN**  
1040               **TEST:**    A call to *mq\_receive()*, when the implementation has detected a data corruption  
1041                           problem with the message, returns a value of -1 and sets *errno* to [EBADMSG].  
1042               **ELSE NO\_TEST\_SUPPORT**  
1043       **ELSE NO\_OPTION**  
1044       Conformance for *mq\_receive*: *PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1045       **15.2.6**    **Notify Process that a Message is Available on a Queue**

1046       Function: *mq\_notify()*

1047       **15.2.6.1**   **Synopsis**

1048       **1**  
1049        *M\_GA\_stdC\_proto\_decl(int; mq\_notify; mqd\_t mqdes, const struct sigevent \*notification;*  
1050        *mqueue.h;;)*  
1051       **SEE:**       Assertion GA\_stdC\_proto\_decl in §2.7.3  
1052       Conformance for *mq\_notify*: *PASS[1, 2], NO\_OPTION*

1053       **2**  
1054        *M\_GA\_commonC\_int\_result\_decl(mq\_notify; mqueue.h;;)*  
1055       **SEE:**       Assertion GA\_commonC\_int\_result\_decl in §2.7.3  
1056       Conformance for *mq\_notify*: *PASS[1, 2], NO\_OPTION*

1057       **3**  
1058        *M\_GA\_macro\_result\_decl(int; mq\_notify; mqueue.h;;)*  
1059       **SEE:**       Assertion GA\_macro\_result\_decl in §1.3.4  
1060       Conformance for *mq\_notify*: *PASS, NO\_OPTION*

1061       **4**  
1062        *M\_GA\_macro\_args (mq\_notify; mqueue.h;;)*  
1063       **SEE:**       Assertion GA\_macro\_args in §2.7.3  
1064       Conformance for *mq\_notify*: *PASS, NO\_OPTION*

1065 **15.2.6.2 Description**1066 **mq\_notify**1067 **IF PCTS\_mq\_notify THEN**

1068 **TEST:** A call to *mq\_notify(mqdes, notification)*, if the argument *notification* is not **NULL**,  
 1069 registers the calling process to be notified of message arrival at an empty message  
 1070 queue associated with the specified message queue descriptor, *mqdes*, and returns a  
 1071 value of zero.

1072 The notification specified by the *notification* argument is sent to the process when the  
 1073 message queue transitions from empty to nonempty.

1074 **ELSE NO\_OPTION**1075 *Conformance for mq\_notify: PASS, NO\_OPTION*1076 **R\_1 IF PCTS\_mq\_notify THEN**

1077 **TEST:** At any time, only one process may be registered for notification by a message queue.  
 1078 When the calling process, or any other process, has already registered for notification  
 1079 of message arrival at the specified message queue, subsequent attempts to register for  
 1080 that message queue fails.

1081 **ELSE NO\_OPTION**1082 **SEE:** Assertion *mq\_notify\_EBUSY* in §15.2.5.41083 **5 IF PCTS\_mq\_notify THEN**

1084 **TEST:** When *notification* is **NULL** and the process currently registered for notification by the  
 1085 specified message queue, the existing registration is removed.

1086 **ELSE NO\_OPTION**1087 *Conformance for mq\_notify: PASS, NO\_OPTION*1088 **6 IF PCTS\_mq\_notify THEN**

1089 **TEST:** When the notification is sent to the registered process, its registration is removed and  
 1090 the message queue is made available for registration.

1091 **ELSE NO\_OPTION**1092 *Conformance for mq\_notify: PASS, NO\_OPTION*1093 **7 IF PCTS\_mq\_notify THEN**1094 **IF PCTS\_mq\_receive THEN**

1095 **TEST:** When a process has registered for notification of message arrival at a message queue,  
 1096 and some process is blocked in *mq\_receive()* waiting to receive a message when a  
 1097 message arrives at the queue, the arriving message satisfies the appropriate  
 1098 *mq\_receive()* (see POSIX.1b {3} §15.2.5).

1099 The resulting behavior is as if the message queue remains empty, and no notification  
 1100 is sent.

1101 **ELSE NO\_TEST\_SUPPORT**1102 **ELSE NO\_OPTION**1103 *Conformance for mq\_notify: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*1104 **D\_1 IF PCTS\_mq\_notify and a PCD.1b documents the following THEN**

1105 **TEST:** A PCD.1b that documents whether or not it supports the *mq\_notify()* function does so  
 1106 in §15.2.6.2.

1107 **ELSE NO\_OPTION**1108 *Conformance for mq\_notify: PASS, NO\_OPTION*1109 **15.2.6.3 Returns**1110 **R\_2 IF PCTS\_mq\_notify THEN**1111 **TEST:** When a call to *mq\_notify()* completes successfully, it returns a value of 0.1112 **ELSE NO\_OPTION**

1113           **SEE:**       Assertion `mq_notify` in §15.2.6.2

1114   **R\_3 IF** `PCTS_mq_notify` **THEN**

1115           **TEST:**     When a call to `mq_notify()` completes unsuccessfully, it returns a value of -1, and sets

1116                       `errno` to indicate the error.

1117           **ELSE** `NO_OPTION`

1118           **SEE:**       All assertions in §15.2.6.4

1119   **15.2.6.4 Errors**

1120   **8 IF** `PCTS_mq_notify` **THEN**

1121           **TEST:**     A call to `mq_notify(mqdes, notification)`, when the `mqdes` argument is not a valid

1122                       message queue descriptor, returns a value of -1 and sets `errno` to `[EBADF]`.

1123           **ELSE** `NO_OPTION`

1124           Conformance for `mq_notify`: `PASS, NO_OPTION`

1125   **mq\_notify\_EBUSY**

1126   **IF** `PCTS_mq_notify` **THEN**

1127           **TEST:**     A call to `mq_notify()`, when a process is already registered for notification by the

1128                       message queue, returns a value of -1 and sets `errno` to `[EBUSY]`.

1129           **ELSE** `NO_OPTION`

1130           Conformance for `mq_notify`: `PASS, NO_OPTION`

1131   **9 IF** not `PCTS_mq_notify` **THEN**

1132           **TEST:**     A call to `mq_notify()` returns a value of -1 and sets `errno` to `[ENOSYS]`.

1133           **ELSE** `NO_OPTION`

1134           Conformance for `mq_notify`: `PASS, NO_OPTION`

1135   **15.2.7 Set Message Queue Attributes**

1136   Function: `mq_setattr()`

1137   **15.2.7.1 Synopsis**

1138

1139   **1**

1140       *`M_GA_stdC_proto_decl(int; mq_setattr; mqd_t mqdes, const struct mq_attr *mqstat, struct mq_attr`*

1141       *`*omqstat; mqueue.h;;)`*

1142       **SEE:**       Assertion `GA_stdC_proto_decl` in §2.7.3

1143       Conformance for `mq_setattr`: `PASS[1, 2], NO_OPTION`

1144   **2**

1145       *`M_GA_commonC_int_result_decl(mq_setattr; mqueue.h;;)`*

1146       **SEE:**       Assertion `GA_commonC_int_result_decl` in §2.7.3

1147       Conformance for `mq_setattr`: `PASS[1, 2], NO_OPTION`

1148   **3**

1149       *`M_GA_macro_result_decl(int; mq_setattr; mqueue.h;;)`*

1150       **SEE:**       Assertion `GA_macro_result_decl` in §1.3.4

1151       Conformance for `mq_setattr`: `PASS, NO_OPTION`

1152   **4**

1153       *`M_GA_macro_args (mq_setattr; mqueue.h;;)`*

1154       **SEE:**       Assertion `GA_macro_args` in §2.7.3

1155       Conformance for `mq_setattr`: `PASS, NO_OPTION`



1156 **15.2.7.2 Description**1157 **mq\_setattr**1158 **IF PCTS\_mq\_setattr THEN**1159 **TEST:** A call to *mq\_setattr()* sets attributes associated with the message queue specified by  
1160 *mqdes* and returns a value of zero.1161 **ELSE NO\_OPTION**1162 *Conformance for mq\_setattr: PASS, NO\_OPTION*1163 **5 IF PCTS\_mq\_setattr THEN**1164 **TEST:** The message queue attributes corresponding to *mq\_flags*, defined in the *mq\_attr*  
1165 structure are set to the specified values upon successful completion of *mq\_setattr()*.1166 **ELSE NO\_OPTION**1167 *Conformance for mq\_setattr: PASS, NO\_OPTION*1168 **D\_1 IF PCTS\_mq\_setattr THEN**1169 **TEST:** The PCD.1b documents any implementation-defined flags that can be set in *mq\_flags*  
1170 in §15.2.7.2.1171 **ELSE NO\_OPTION**1172 *Conformance for mq\_setattr: PASS, NO\_OPTION*1173 **6 IF PCTS\_mq\_setattr THEN**1174 **TEST:** The values of the *mq\_maxmsg*, *mq\_msgsize*, and *mq\_curmsgs* members of the *mq\_attr*  
1175 structure are ignored by *mq\_setattr()*.1176 **ELSE NO\_OPTION**1177 *Conformance for mq\_setattr: PASS, NO\_OPTION*1178 **7 IF PCTS\_mq\_setattr THEN**1179 **IF PCTS\_mq\_getattr THEN**1180 **TEST:** When *omqstat* is non-NULL, the function *mq\_setattr()* stores, in the location  
1181 referenced by *omqstat*, the previous message queue attributes and the current  
1182 queue status.1183 These values are the same as would be returned by a call to *mq\_getattr()* at that  
1184 point.1185 **ELSE NO\_TEST\_SUPPORT**1186 **ELSE NO\_OPTION**1187 *Conformance for mq\_setattr: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*1188 **D\_2 IF PCTS\_mq\_setattr and a PCD.1b documents the following THEN**1189 **TEST:** A PCD.1b that documents whether or not it supports the *mq\_setattr()* function does so  
1190 in §15.2.7.2.1191 **ELSE NO\_OPTION**1192 *Conformance for mq\_setattr: PASS, NO\_OPTION*1193 **15.2.7.3 Returns**1194 **R\_1 IF PCTS\_mq\_setattr THEN**1195 **TEST:** When a call to *mq\_setattr()* completes successfully, the function returns a value of 0  
1196 and changes the attributes of the message queue as specified.1197 **ELSE NO\_OPTION**1198 **SEE:** Assertion *mq\_setattr* in §15.2.7.21199 **R\_2 IF PCTS\_mq\_setattr THEN**1200 **TEST:** When a call to *mq\_setattr()* completes unsuccessfully, the message queue attributes  
1201 are unchanged, and the function returns a value of -1 and sets *errno* to indicate the  
1202 error.1203 **ELSE NO\_OPTION**

1204           **SEE:**       All assertions in §15.2.7.4

1205    **15.2.7.4   Errors**

1206    **9        IF** not *PCTS\_mq\_setattr* **THEN**

1207           **TEST:**     A call to *mq\_setattr*() returns a value of -1 and sets *errno* to [ENOSYS].

1208           **ELSE** *NO\_OPTION*

1209           *Conformance for mq\_setattr: PASS, NO\_OPTION*

1210    **15.2.8    Get Message Queue Attributes**

1211    Function: *mq\_getattr*()

1212    **15.2.8.1   Synopsis**

1213    **1**

1214           *M\_GA\_stdC\_proto\_decl(int; mq\_getattr; mqd\_t mqdes, struct mq\_attr \*mqstat; mqueue.h;;)*

1215           **SEE:**       Assertion *GA\_stdC\_proto\_decl* in §2.7.3

1216           *Conformance for mq\_getattr: PASS[1, 2], NO\_OPTION*

1217    **2**

1218           *M\_GA\_commonC\_int\_result\_decl(mq\_getattr; mqueue.h;;)*

1219           **SEE:**       Assertion *GA\_commonC\_int\_result\_decl* in §2.7.3

1220           *Conformance for mq\_getattr: PASS[1, 2], NO\_OPTION*

1221    **3**

1222           *M\_GA\_macro\_result\_decl(int; mq\_getattr; mqueue.h;;)*

1223           **SEE:**       Assertion *GA\_macro\_result\_decl* in §1.3.4

1224           *Conformance for mq\_getattr: PASS, NO\_OPTION*

1225    **4**

1226           *M\_GA\_macro\_args ( mq\_getattr; mqueue.h;;)*

1227           **SEE:**       Assertion *GA\_macro\_args* in §2.7.3

1228           *Conformance for mq\_getattr: PASS, NO\_OPTION*

1229    **15.2.8.2   Description**

1230    **mq\_getattr**

1231           **IF** *PCTS\_mq\_getattr* **THEN**

1232               **TEST:**     A call to *mq\_getattr*() gets status information and attributes associated with the

1233                           message queue specified in *mqdes* and returns zero.

1234           **ELSE** *NO\_OPTION*

1235           *Conformance for mq\_getattr: PASS, NO\_OPTION*

1236    **5        IF** *PCTS\_mq\_getattr* **THEN**

1237               **IF** *PCTS\_mq\_setattr* **THEN**

1238                   **TEST:**     After a successful call to *mq\_getattr*(*mqdes, mqstat*) the *mq\_flags* member within the

1239                           *mq\_attr* structure referenced by the *mqstat* argument has the value that was set when

1240                           the message queue was created but with modifications made by subsequent

1241                           *mq\_setattr*() calls.

1242               **ELSE** *NO\_TEST\_SUPPORT*

1243           **ELSE** *NO\_OPTION*

1244           *Conformance for mq\_getattr: PASS, NO\_TEST\_SUPPORT, NO\_OPTION*

1245    **6        IF** *PCTS\_mq\_getattr* **THEN**

1246                   **TEST:**     After a successful call to *mq\_getattr(mqdes, mqstat)* the *mq\_maxmsg* and the  
 1247                                   *mq\_msgsize* members within the *mq\_attr* structure referenced by the *mqstat* argument  
 1248                                   have the values that were set at message queue creation.  
 1249                   **ELSE NO\_OPTION**  
 1250                   *Conformance for mq\_getattr: PASS, NO\_OPTION*

1251       **7**           **IF** *PCTS\_mq\_getattr* **THEN**  
 1252                   **TEST:**     After a successful call to *mq\_getattr(mqdes, mqstat)* the *mq\_curmsgs* member within  
 1253                                   the *mq\_attr* structure referenced by the *mqstat* argument has the value that is set  
 1254                                   according to the number of messages currently on the queue.  
 1255                   **ELSE NO\_OPTION**  
 1256                   *Conformance for mq\_getattr: PASS, NO\_OPTION*

1257       **D\_1**       **IF** *PCTS\_mq\_getattr* and a PCD.1b documents the following **THEN**  
 1258                   **TEST:**     A PCD.1b that documents whether or not it supports the *mq\_getattr()* function does so  
 1259                                   in §15.2.8.2.  
 1260                   **ELSE NO\_OPTION**  
 1261                   *Conformance for mq\_getattr: PASS, NO\_OPTION*

1262       **15.2.8.3 Returns**

1263       **R\_1** **IF** *PCTS\_mq\_getattr* **THEN**  
 1264                   **TEST:**     When a call to *mq\_getattr()* completes successfully, it returns 0.  
 1265                   **ELSE NO\_OPTION**  
 1266                   **SEE:**     Assertion *mq\_getattr* in §15.2.8.2

1267       **R\_2** **IF** *PCTS\_mq\_getattr* **THEN**  
 1268                   **TEST:**     When a call to *mq\_getattr()* completes unsuccessfully, the function returns -1 and sets  
 1269                                   *errno* to indicate the error.  
 1270                   **ELSE NO\_OPTION**  
 1271                   **SEE:**     All assertions in §15.2.8.4

1272       **8**           **IF** *PCTS\_mq\_getattr* **THEN**  
 1273                   **TEST:**  
 1274                   **ELSE NO\_OPTION**  
 1275                   *Conformance for mq\_getattr: PASS, NO\_OPTION*

1276       **15.2.8.4 Errors**

1277       **9**           **IF** *PCTS\_mq\_getattr* **THEN**  
 1278                   **TEST:**     A call to *mq\_getattr(mqdes, mqstat)*, when the *mqdes* argument is not a valid message  
 1279                                   queue descriptor, returns a value of -1 and sets *errno* to [EBADF].  
 1280                   **ELSE NO\_OPTION**  
 1281                   *Conformance for mq\_getattr: PASS, NO\_OPTION*

1282       **10**       **IF** not *PCTS\_mq\_getattr* **THEN**  
 1283                   **TEST:**     A call to *mq\_getattr()* returns a value of -1 and sets *errno* to [ENOSYS].  
 1284                   **ELSE NO\_OPTION**  
 1285                   *Conformance for mq\_getattr: PASS, NO\_OPTION*

## Annex A (normative)

### Conforming Test Results

#### 180 **A.1 General**

##### 181 **Assertions for conformance**

182	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
183	1.3.3.3	GD_CommonC_diffs	General Documentation Assertion, no test results
184	1.3.4	GA_macro_args	General Assertion, no test results
185	1.3.4	GA_macro_result_decl	General Assertion, no test results

#### 186 **A.2 Terminology and General Requirements**

##### 187 **Assertions for definitions**

188	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
189	2.2.2.40	GA_portableFileNames	General Assertion, no test results
190	2.2.2.40	GA_upperLowerNames	General Assertion, no test results
191	2.2.2.105	D_1	PASS
192	2.2.2.105	D_2	PASS
193	2.2.2.108	R_1	Reference Assertion, no test results
194	2.2.2.109	R_2	Reference Assertion, no test results
195	2.2.2.119	GA_syncIODataIntegrityRead	General Assertion, no test results
196	2.2.2.119	GA_syncIODataIntegrityWbeforeR	General Assertion, no test results
197	2.2.2.119	GA_syncIODataIntegrityWrite	General Assertion, no test results
198	2.2.2.120	GA_syncIOFileIntegrityRead	General Assertion, no test results
199	2.2.2.120	GA_syncIOFileIntegrityWrite	General Assertion, no test results
200	2.2.2.126	D_3	PASS

##### 201 **Assertions for General Concepts**

202	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
203	2.3.2	GA_AP_overrideFileAccess	General Assertion, no test results
204	2.3.2	GA_AP_overrideExecAccess	General Assertion, no test results
205	2.3.2	GA_AP_classAccess	General Assertion, no test results

206	2.3.2	GA_AdditionalAccessControl	General Assertion, no test results
207	2.3.2	GA_AlternateAccessControl	General Assertion, no test results
208	2.3.2	GA_AltAccessEnable	General Assertion, no test results
209	2.3.2	GA_AltAccessDisable	General Assertion, no test results
210	2.3.5	GA_statTimeUpdate	General Assertion, no test results
211	2.3.5	GA_NoOpenTimeUpdate	General Assertion, no test results
212	2.3.5	GA_NoROFSTimeUPDATE	General Assertion, no test results
213	2.3.6	GA_PRDot	General Assertion, no test results
214	2.3.6	GA_PRSlash	General Assertion, no test results
215	2.3.6	GA_PR3SLASH	General Assertion, no test results
216	2.3.6	GA_PRDotDot	General Assertion, no test results
217	2.3.6	GA_PRRelativeSlash	General Assertion, no test results
218	2.3.6	GA_PRRelativeSlashSlash	General Assertion, no test results
219	2.3.6	GA_PRRenameRelativeSlashSlash	General Assertion, no test results
220	2.3.6	GA_PRRelativeCWD	General Assertion, no test results
221	2.3.6	GA_PRRelativeDotCWD	General Assertion, no test results
222	2.3.6	GA_PRRelativeDotDotCWD	General Assertion, no test results
223	2.3.6	GA_PRRelativeSlashSlashCWD	General Assertion, no test results
224	2.3.6	GA_PRNoTrunc	General Assertion, no test results
225	2.3.6	GA_PRNoTruncError	General Assertion, no test results

#### 226 **Assertions for Error Numbers**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
227			
228	2.4	2	PASS
229	2.4	GD_OptionalErrors	General Documentation Assertion, no test results
230	2.4	GA_Optional ErrorsUndetected	General Assertion, no test results

#### 231 **Assertions for Environment Description**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
232			
233	2.6	GA_ExecNoSlash	General Assertion, no test results
234	2.6	GA_ExecColon	General Assertion, no test results
235	2.6	GA_ExecInsertSlash	General Assertion, no test results
236	2.6	GA_ExecTwoColons	General Assertion, no test results
237	2.6	GA_ExecInitialColon	General Assertion, no test results
238	2.6	GA_ExecTrailingColon	General Assertion, no test results
239	2.6	GA_ExecPathSearchOrder	General Assertion, no test results
240	2.6	GA_ExecPathSearchOrder	General Assertion, no test results
241	2.6	GA_ExecPathSearchOrder	General Assertion, no test results

#### 242 **Assertions for C Language Definitions**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
243			
244	2.7.1	4	PASS
245	2.7.1	4.1	PASS
246	2.7.2	5	PASS, NO_OPTION, NO_TEST
247	2.7.2	6	PASS, NO_OPTION, NO_TEST

248	2.7.2	D_1	PASS, NO_OPTION
249	2.7.2.1	7	PASS, NO_TEST
250	2.7.2.1	8	PASS, NO_TEST
251	2.7.2.2	9	PASS, NO_TEST
252	2.7.3	GastdC_proto_decl	General Assertion, no test results
253	2.7.3	GA_commonC_result_decl	General Assertion, no test results
254	2.7.3	GA_commonC_int_result_decl	General Assertion, no test results
255	2.7.3	GA_setjmpDecl	General Assertion, no test results
256	2.7.3	GA_sigsetjmpDecl	General Assertion, no test results
257	2.7.3	GA_macro_args	General Assertion, no test results
258	2.7.3	D_2	PASS, NO_OPTION

### 259 Assertions for Numerical Limits

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
260			
261	2.8.2	2	PASS
262	2.8.4	4	PASS
263	2.8.4	5	PASS, NO_OPTION
264	2.8.4	D_2	PASS
265	2.8.7	18	PASS

### 266 Assertions for Symbolic Constants

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
267			
268	2.9	D_1	PASS
269	2.9	D_2	PASS
270	2.9.3	1	PASS
271	2.9.3	2	PASS
272	2.9.3	3	PASS, NO_OPTION
273	2.9.3	4	PASS, NO_OPTION
274	2.9.3	5	PASS, NO_OPTION
275	2.9.4	6	PASS
276	2.9.4	7	PASS, NO_TEST
277	2.9.4	8	PASS, NO_TEST

## 278 A.3 Process Primitives

### 279 Assertions for fork

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
280			
281	3.1.1.2	1	PASS, NO_OPTION
282	3.1.1.2	2	PASS, NO_OPTION, NO_TEST_SUPPORT
283	3.1.1.2	3	PASS, NO_OPTION
284	3.1.1.2	4	PASS, NO_OPTION
285	3.1.1.2	5	PASS, NO_OPTION
286	3.1.1.2	6	PASS, NO_OPTION
287	3.1.1.2	7	PASS, NO_OPTION
288	3.1.1.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
289	3.1.1.2	9	PASS, NO_OPTION

290	3.1.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
291	3.1.1.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
292	3.1.1.2	12	PASS, NO_OPTION, NO_TEST
293	3.1.1.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT

---

294 **Assertions for execl**

295	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
296	3.1.2.2	1	PASS, NO_OPTION
297	3.1.2.2	2	PASS, NO_OPTION
298	3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
299	3.1.2.2	D_1	PASS, NO_OPTION
300	3.1.2.2	4	PASS, NO_OPTION, NO_TEST
301	3.1.2.2	5	PASS, NO_OPTION
302	3.1.2.2	D_2	PASS, NO_OPTION
303	3.1.2.2	6	PASS, NO_OPTION, NO_TEST
304	3.1.2.2	7	PASS, NO_OPTION
305	3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
306	3.1.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
307	3.1.2.2	10	PASS, NO_OPTION, NO_TEST
308	3.1.2.2	D_3	PASS, NO_OPTION
309	3.1.2.2	11	PASS, NO_OPTION, NO_TEST
310	3.1.2.2	D_4	PASS, NO_OPTION

---

311 **Assertions for execv**

312	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
313	3.1.2.2	1	PASS, NO_OPTION
314	3.1.2.2	2	PASS, NO_OPTION
315	3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
316	3.1.2.2	D_1	PASS, NO_OPTION
317	3.1.2.2	4	PASS, NO_OPTION, NO_TEST
318	3.1.2.2	5	PASS, NO_OPTION
319	3.1.2.2	D_2	PASS, NO_OPTION
320	3.1.2.2	6	PASS, NO_OPTION, NO_TEST
321	3.1.2.2	7	PASS, NO_OPTION
322	3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
323	3.1.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
324	3.1.2.2	10	PASS, NO_OPTION, NO_TEST
325	3.1.2.2	D_3	PASS, NO_OPTION
326	3.1.2.2	11	PASS, NO_OPTION, NO_TEST
327	3.1.2.2	D_4	PASS, NO_OPTION

---

328 **Assertions for execl**

329	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
330	3.1.2.2	1	PASS, NO_OPTION
331	3.1.2.2	2	PASS, NO_OPTION
332	3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT

333	3.1.2.2	D_1	PASS, NO_OPTION
334	3.1.2.2	4	PASS, NO_OPTION, NO_TEST
335	3.1.2.2	5	PASS, NO_OPTION
336	3.1.2.2	D_2	PASS, NO_OPTION
337	3.1.2.2	6	PASS, NO_OPTION, NO_TEST
338	3.1.2.2	7	PASS, NO_OPTION
339	3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
340	3.1.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
341	3.1.2.2	10	PASS, NO_OPTION, NO_TEST
342	3.1.2.2	D_3	PASS, NO_OPTION
343	3.1.2.2	11	PASS, NO_OPTION, NO_TEST
344	3.1.2.2	D_4	PASS, NO_OPTION

---



345 **Assertions for execlp**

346	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
347	3.1.2.2	1	PASS, NO_OPTION
348	3.1.2.2	2	PASS, NO_OPTION
349	3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
350	3.1.2.2	D_1	PASS, NO_OPTION
351	3.1.2.2	4	PASS, NO_OPTION, NO_TEST
352	3.1.2.2	5	PASS, NO_OPTION
353	3.1.2.2	D_2	PASS, NO_OPTION
354	3.1.2.2	6	PASS, NO_OPTION, NO_TEST
355	3.1.2.2	7	PASS, NO_OPTION
356	3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
357	3.1.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
358	3.1.2.2	10	PASS, NO_OPTION, NO_TEST
359	3.1.2.2	D_3	PASS, NO_OPTION
360	3.1.2.2	11	PASS, NO_OPTION, NO_TEST
361	3.1.2.2	D_4	PASS, NO_OPTION

362 **Assertions for execvp**

363	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
364	3.1.2.2	1	PASS, NO_OPTION
365	3.1.2.2	2	PASS, NO_OPTION
366	3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
367	3.1.2.2	D_1	PASS, NO_OPTION
368	3.1.2.2	4	PASS, NO_OPTION, NO_TEST
369	3.1.2.2	5	PASS, NO_OPTION
370	3.1.2.2	D_2	PASS, NO_OPTION
371	3.1.2.2	6	PASS, NO_OPTION, NO_TEST
372	3.1.2.2	7	PASS, NO_OPTION
373	3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
374	3.1.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
375	3.1.2.2	10	PASS, NO_OPTION, NO_TEST
376	3.1.2.2	D_3	PASS, NO_OPTION
377	3.1.2.2	11	PASS, NO_OPTION, NO_TEST
378	3.1.2.2	D_4	PASS, NO_OPTION

379 **Assertions for Process Termination**

380	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
381	No assertions for Process Termination		

382 **Assertions for wait**

383	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
384	No assertions for wait		

385 **Assertions for \_exit**

386	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
387	3.2.2.2	1	PASS, NO_OPTION, NO_TEST_SUPPORT
388	3.2.2.2	2	PASS, NO_OPTION, NO_TEST_SUPPORT
389	3.2.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
390	3.2.2.2	4	PASS, NO_OPTION, NO_TEST
391	3.2.2.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
392	3.2.2.2	6	PASS, NO_OPTION, NO_TEST
393	3.2.2.2	D_1	PASS, NO_OPTION
394	3.2.2.2	D_2	PASS

395 **Assertions for signal.h**

396	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
397	3.3.1.1	1	PASS
398	3.3.1.1	2	PASS, NO_OPTION
399	3.3.1.1	3	PASS
400	3.3.1.1	4	PASS
401	3.3.1.1	5	PASS
402	3.3.1.1	D_1	PASS
403	3.3.1.2	6	PASS
404	3.3.1.2	D_2	PASS, NO_OPTION
405	3.3.1.2	7	PASS
406	3.3.1.2	D_3	PASS, NO_OPTION
407	3.3.1.2	GA_sigev_value	General Assertion, no test results
408	3.3.1.2	GA_sigqueueValue	General Assertion, no test results
409	3.3.1.2	GA_sigPending Queued	General Assertion, no test results
410	3.3.1.2	D_4	PASS, NO_OPTION
411	3.3.1.2	ga_queuedAndRegularSignals	General Assertion, no test results
412	3.3.1.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
413	3.3.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
414	3.3.1.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
415	3.3.1.3	12	PASS, NO_OPTION, NO_TEST_SUPPORT
416	3.3.1.3	13	PASS, NO_OPTION
417	3.3.1.3	14	PASS, NO_OPTION, NO_TEST
418	3.3.1.3	D_5	PASS, NO_OPTION
419	3.3.1.3	15	PASS, NO_OPTION, NO_TEST_SUPPORT
420	3.3.1.3	16	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
421	3.3.1.3	17	PASS, NO_OPTION, NO_TEST_SUPPORT`

422	3.3.1.3	18	PASS, NO_OPTION
423	3.3.1.3	19	PASS, NO_OPTION
424	3.3.1.3	20	PASS
425	3.3.1.3	21	PASS, NO_OPTION
426	3.3.1.3	22	PASS
427	3.3.1.3	D_6	PASS, NO_OPTION
428	3.3.1.3	23	PASS, NO_OPTION
429	3.3.1.3	24	PASS, NO_OPTION, NO_TEST_SUPPORT
430	3.3.1.3	25	PASS, NO_OPTION
431	3.3.1.3	26	PASS, NO_OPTION
432	3.3.1.3	D_7	PASS
433	3.3.1.3	27	PASS, NO_OPTION
434	3.3.1.3	28	PASS, NO_OPTION, NO_TEST_SUPPORT
435	3.3.1.3	29	PASS, NO_OPTION, NO_TEST_SUPPORT
436	3.3.1.3	30	PASS, NO_OPTION
437	3.3.1.3	31	PASS, NO_OPTION
438	3.3.1.3	D_8	PASS, NO_OPTION
439	3.3.1.3	D_9	PASS, NO_OPTION
440	3.3.1.3	32	PASS, NO_OPTION
441	3.3.1.3	33	PASS, NO_OPTION
442	3.3.1.3	34	PASS, NO_OPTION
443	3.3.1.3	35	PASS, NO_OPTION
444	3.3.1.3	36	PASS, NO_OPTION
445	3.3.1.3	37	PASS, NO_OPTION
446	3.3.1.3	38	PASS, NO_OPTION
447	3.3.1.3	39	PASS, NO_OPTION
448	3.3.1.3	40	PASS, NO_OPTION
449	3.3.1.3	41	PASS, NO_OPTION

---

450 **Assertions for kill**

451	Subclause	Assertion ID	Conforming Results
452	No assertions for kill		

---

453 **Assertions for sigemptyset**

454	Subclause	Assertion ID	Conforming Results
455	No assertions for sigemptyset		

---

456 **Assertions for sigfillset**

457	Subclause	Assertion ID	Conforming Results
458	No assertions for sigfillset		

---

459 **Assertions for sigaddset**

460	Subclause	Assertion ID	Conforming Results
461	No assertions for sigaddset		

---

462 **Assertions for sigdelset**

463	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
464	No assertions for sigdelset		

465 **Assertions for sigismember**

466	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
467	No assertions for sigismember		

468 **Assertions for sigaction**

469	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
470	3.3.4.2	1	PASS
471	3.3.4.2	2	PASS, NO_OPTION
472	3.3.4.2	3	PASS
473	3.3.4.2	4	PASS
474	3.3.4.2	D_1	PASS
475	3.3.4.2	5	PASS, NO_OPTION
476	3.3.4.4	6	PASS, NO_OPTION

477 **Assertions for sigprocmask**

478	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
479	3.3.5.2	D_1	PASS, NO_OPTION

480 **Assertions for sigpending**

481	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
482	No assertions for sigpending		

483 **Assertions for sigsuspend**

484	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
485	No assertions for sigsuspend		

486 **Assertions for sigwaitinfo**

487	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
488	3.3.8.1	1	PASS[1,2], NO_OPTION
489	3.3.8.1	2	PASS[1,2], NO_OPTION
490	3.3.8.1	3	PASS, NO_OPTION
491	3.3.8.1	4	PASS, NO_OPTION
492	3.3.8.2	9	PASS, NO_OPTION
493	3.3.8.2	10	PASS, NO_OPTION
494	3.3.8.2	D_1	PASS, NO_OPTION
495	3.3.8.2	11	PASS, NO_OPTION
496	3.3.8.2	12	PASS, NO_OPTION
497	3.3.8.2	13	PASS, NO_OPTION
498	3.3.8.2	14	PASS, NO_OPTION, NO_TEST
499	3.3.8.2	D_2	PASS, NO_OPTION
500	3.3.8.2	15	PASS, NO_OPTION
501	3.3.8.2	18	PASS, NO_OPTION

502	3.3.8.2	D_4	PASS, NO_OPTION
503	3.3.8.3	R_1	Reference Assertion, no test results
504	3.3.8.3	R_2	Reference Assertion, no test results
505	3.3.8.4	19	PASS, NO_OPTION
506	3.3.8.4	20	PASS, NO_OPTION
507	3.3.8.4	22	PASS, NO_OPTION
508	3.3.8.4	D_5	PASS, NO_OPTION

---

509 **Assertions for sigtimedwait**

510	Subclause	Assertion ID	Conforming Results
511	3.3.8.1	5	PASS[5, 6], NO_OPTION
512	3.3.8.1	6	PASS[5, 6], NO_OPTION
513	3.3.8.1	7	PASS, NO_OPTION
514	3.3.8.1	8	PASS, NO_OPTION
515	3.3.8.1	8	PASS, NO_OPTION
516	3.3.8.2	10	PASS, NO_OPTION
517	3.3.8.2	D_1	PASS, NO_OPTION
518	3.3.8.2	11	PASS, NO_OPTION
519	3.3.8.2	12	PASS, NO_OPTION
520	3.3.8.2	13	PASS, NO_OPTION
521	3.3.8.2	14	PASS, NO_OPTION, NO_TEST
522	3.3.8.2	D_2	PASS, NO_OPTION
523	3.3.8.2	15	PASS, NO_OPTION
524	3.3.8.2	16	PASS, NO_OPTION
525	3.3.8.2	17	PASS, NO_OPTION
526	3.3.8.2	D_3	PASS, NO_OPTION
527	3.3.8.2	18	PASS, NO_OPTION
528	3.3.8.2	D_4	PASS, NO_OPTION
529	3.3.8.4	19	PASS, NO_OPTION
530	3.3.8.4	20	PASS, NO_OPTION
531	3.3.8.4	21	PASS, NO_OPTION

---

532 **Assertions for sigqueue**

533	Subclause	Assertion ID	Conforming Results
534	3.3.9.1	1	PASS[1, 2], NO_OPTION
535	3.3.9.1	2	PASS[1, 2], NO_OPTION
536	3.3.9.1	3	PASS, NO_OPTION
537	3.3.9.1	4	PASS, NO_OPTION
538	3.3.9.2	5	PASS, NO_OPTION
539	3.3.9.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
540	3.3.9.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
541	3.3.9.2	8	PASS, NO_OPTION
542	3.3.9.2	9	PASS, NO_OPTION
543	3.3.9.2	10	PASS, NO_OPTION
544	3.3.9.2	11	PASS, NO_OPTION
545	3.3.9.2	D_1	PASS, NO_OPTION
546	3.3.9.2	12	PASS, NO_OPTION

547	3.3.9.2	D_2	PASS, NO_OPTION
548	3.3.9.3	R_1	Reference Assertion, no test results
549	3.3.9.3	R_2	Reference Assertion, no test results
550	3.3.9.4	13	PASS, NO_OPTION, NO_TEST
551	3.3.9.4	14	PASS, NO_OPTION
552	3.3.9.4	15	PASS, NO_OPTION, NO_TEST
553	3.3.9.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
554	3.3.9.4	17	PASS, NO_OPTION
555	3.3.9.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT
556	3.3.9.4	19	PASS, NO_OPTION

557 **Assertions for intro\_timer\_ops**

558	Subclause	Assertion ID	Conforming Results
559	No assertions for intro_timer_ops		

560 **Assertions for alarm**

561	Subclause	Assertion ID	Conforming Results
562	No assertions for alarm		

563 **Assertions for pause**

564	Subclause	Assertion ID	Conforming Results
565	No assertions for pause		

566 **Assertions for sleep**

567	Subclause	Assertion ID	Conforming Results
568	No assertions for sleep		

569 **A.4 Process Environment**

570 **Assertions for sysconf**

571	Subclause	Assertion ID	Conforming Results
572	4.8.1.2	1	PASS
573	4.8.1.2	2	PASS, NO_TEST_SUPPORT
574	4.8.1.2	3	PASS, NO_TEST_SUPPORT
575	4.8.1.2	4	PASS, NO_TEST_SUPPORT
576	4.8.1.2	5	PASS, NO_TEST_SUPPORT

577 **A.5 Files and Directories**

578 **Assertions for intro5**

579	Subclause	Assertion ID	Conforming Results
580	No assertions for sleep		

581 **Assertions for dirent.h**

Subclause	Assertion ID	Conforming Results
No assertions for dirent.h		

584 **Assertions for directory operations**

Subclause	Assertion ID	Conforming Results
No assertions for directory operations		

587 **Assertions for chdir**

Subclause	Assertion ID	Conforming Results
No assertions for chdir		

590 **Assertions for getcwd**

Subclause	Assertion ID	Conforming Results
No assertions for getcwd		

593 **Assertions for open**

Subclause	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

598 **Assertions for creat**

Subclause	Assertion ID	Conforming Results
No assertions for creat		

601 **Assertions for umask**

Subclause	Assertion ID	Conforming Results
No assertions for umask		

604 **Assertions for link**

Subclause	Assertion ID	Conforming Results
No assertions for link		

607 **Assertions for mkdir**

Subclause	Assertion ID	Conforming Results
No assertions for mkdir		

610 **Assertions for mkfifo**

Subclause	Assertion ID	Conforming Results
No assertions for mkfifo		

613 **Assertions for unlink**

614	Subclause	Assertion ID	Conforming Results
615	No assertions for unlink		

616 **Assertions for rmdir**

617	Subclause	Assertion ID	Conforming Results
618	No assertions for rmdir		

619 **Assertions for rename**

620	Subclause	Assertion ID	Conforming Results
621	No assertions for rename		

622 **Assertions for stat.h**

623	Subclause	Assertion ID	Conforming Results
624	5.6.1.1	D_1	PASS, NO_OPTION
625	5.6.1.1	1	PASS
626	5.6.1.1	2	PASS, NO_TEST_SUPPORT
627	5.6.1.1	3	PASS, NO_TEST_SUPPORT
628	5.6.1.1	4	PASS, NO_TEST_SUPPORT
629	5.6.1.1	5	PASS, NO_TEST_SUPPORT
630	5.6.1.1	6	PASS, NO_TEST_SUPPORT
631	5.6.1.1	7	PASS, NO_TEST_SUPPORT
632	5.6.1.1	8	PASS, NO_TEST_SUPPORT
633	5.6.1.1	9	PASS, NO_TEST_SUPPORT
634	5.6.1.1	10	PASS, NO_TEST_SUPPORT

635 **Assertions for fstat**

636	Subclause	Assertion ID	Conforming Results
637	5.6.2.2	1	PASS

638 **Assertions for stat**

639	Subclause	Assertion ID	Conforming Results
640	No assertions for stat		

641 **Assertions for access**

642	Subclause	Assertion ID	Conforming Results
643	No assertions for access		

644 **Assertions for fchmod**

645	Subclause	Assertion ID	Conforming Results
646	5.6.4.1	1	PASS[1, 2], NO_OPTION
647	5.6.4.1	2	PASS[1, 2], NO_OPTION
648	5.6.4.1	3	PASS, NO_OPTION
649	5.6.4.1	4	PASS, NO_OPTION
650	5.6.4.2	5	PASS, NO_OPTION
651	5.6.4.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
652	5.6.4.2	7	PASS, NO_OPTION



653	5.6.4.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
654	5.6.4.2	9	PASS, NO_OPTION
655	5.6.4.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
656	5.6.4.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
657	5.6.4.3	R_1	Reference Assertion, no test results
658	5.6.4.3	R_2	Reference Assertion, no test results
659	5.6.4.4	14	PASS, NO_OPTION
660	5.6.4.4	15	PASS, NO_OPTION
661	5.6.4.4	16	PASS, NO_OPTION
662	5.6.4.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT
663	5.6.4.4	18	PASS, NO_OPTION

664 **Assertions for chmod**

665	Subclause	Assertion ID	Conforming Results
666	5.6.4.2	12	PASS, NO_TEST_SUPPORT

667 **Assertions for utime**

668	Subclause	Assertion ID	Conforming Results
669	No assertions for utime		

670 **Assertions for ftruncate**

671	Subclause	Assertion ID	Conforming Results
672	5.6.7.1	1	PASS[1, 2], NO_OPTION
673	5.6.7.1	2	PASS[1, 2], NO_OPTION
674	5.6.7.1	3	PASS, NO_OPTION
675	5.6.7.1	4	PASS, NO_OPTION
676	5.6.7.2	5	PASS, NO_OPTION
677	5.6.7.2	D_1	PASS, NO_OPTION
678	5.6.7.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
679	5.6.7.2	7	PASS, NO_OPTION
680	5.6.7.2	D_2	PASS, NO_OPTION
681	5.6.7.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
682	5.6.7.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT I
683	5.6.7.2	10	PASS, NO_OPTION
684	5.6.7.2	11	PASS, NO_OPTION
685	5.6.7.2	12	PASS, NO_OPTION
686	5.6.7.3	R_1	Reference Assertion, no test results
687	5.6.7.3	R_2	Reference Assertion, no test results
688	5.6.7.4	13	PASS, NO_OPTION
689	5.6.7.4	14	PASS, NO_OPTION
690	5.6.7.4	15	PASS, NO_OPTION

691 **Assertions for pathconf**

692	Subclause	Assertion ID	Conforming Results
693	5.7.1.2	1	PASS

694	5.7.1.2	2	PASS, NO_OPTION
695	5.7.1.2	3	PASS
696	5.7.1.2	4	PASS, NO_OPTION
697	5.7.1.2	5	PASS
698	5.7.1.2	6	PASS, NO_OPTION
699	5.7.1.2	7	PASS
700	5.7.1.2	D_1	PASS, NO_OPTION

701 **Assertions for fpathconf**

702	Subclause	Assertion ID	Conforming Results
703	5.7.1.2	2	PASS, NO_OPTION
704	5.7.1.2	3	PASS
705	5.7.1.2	4	PASS, NO_OPTION
706	5.7.1.2	5	PASS
707	5.7.1.2	6	PASS, NO_OPTION
708	5.7.1.2	7	PASS
709	5.7.1.2	D_1	PASS, NO_OPTION

710 **A.6 Input and Output Primitives**

711 **Assertions for intro6**

712	Subclause	Assertion ID	Conforming Results
713	No assertions for intro6		

714 **Assertions for pipe**

715	Subclause	Assertion ID	Conforming Results
716	No assertions for pipe		

717 **Assertions for dup**

718	Subclause	Assertion ID	Conforming Results
719	No assertions for dup		

720 **Assertions for dup2**

721	Subclause	Assertion ID	Conforming Results
722	No assertions for dup2		

723 **Assertions for close**

724	Subclause	Assertion ID	Conforming Results
725	6.3.1.2	1	PASS, NO_OPTION, NO_TEST
726	6.3.1.2	2	PASS, NO_OPTION, NO_TEST
727	6.3.1.2	D_1	PASS
728	6.3.1.2	3	PASS, NO_OPTION
729	6.3.1.2	4	PASS, NO_OPTION

730 **Assertions for read**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
731			
732	6.4.1.2	1	PASS, NO_OPTION, NO_TEST
733	6.4.1.2	2	PASS, NO_OPTION, NO_TEST
734	6.4.1.2	3	PASS, NO_OPTION, NO_TEST
735	6.4.1.2	D_1	PASS, NO_OPTIONS

736 **Assertions for write**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
737			
738	6.4.2.2	1	PASS, NO_OPTION, NO_TEST
739	6.4.2.2	2	PASS, NO_OPTION, NO_TEST
740	6.4.2.2	D_1	PASS, NO_OPTION

741 **Assertions for fcntl.h**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
742			
743	6.5.1	1	PASS

744 **Assertions for fcntl**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
745			
746	6.5.2.2	4	PASS
747	6.5.2.2	5	PASS
748	6.5.2.2	6	PASS
749	6.5.2.2	7	PASS
750	6.5.2.2	8	PASS
751	6.5.2.2	9	PASS
752	6.5.2.2	10	PASS
753	6.5.2.2	D_1	PASS, NO_OPTION
754	6.5.2.4	41	PASS
755	6.5.2.4	42	PASS, NO_TEST_SUPPORT
756	6.5.2.4	42.1	PASS, NO_TEST_SUPPORT
757	6.5.2.4	42.2	PASS, NO_OPTION
758	6.5.2.4	48	PASS, NO_TEST_SUPPORT
759	6.5.2.4	49	PASS, NO_TEST_SUPPORT
760	6.5.2.4	49.1	PASS, NO_TEST_SUPPORT

761 **Assertions for lseek**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
762			
763	6.5.3.2	D_1	PASS, NO_OPTION

764 **Assertions for syncio**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
765			
766	6.6	D_1	PASS

767 **Assertions for fsync**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
768			
769	6.6.1.1	1	PASS[1, 2], NO_OPTION
770	6.6.1.1	2	PASS[1, 2], NO_OPTION
771	6.6.1.1	3	PASS, NO_OPTION
772	6.6.1.1	4	PASS, NO_OPTION
773	6.6.1.2	D_1	PASS, NO_OPTION
774	6.6.1.2	5	PASS, NO_OPTION, NO_TEST
775	6.6.1.2	R_1	Reference Assertion, no test results
776	6.6.1.2	D_2	PASS, NO_OPTION
777	6.6.1.2	6	PASS, NO_OPTION, NO_TEST
778	6.6.1.2	7	PASS, NO_OPTION, NO_TEST
779	6.6.1.3	R_2	Reference Assertion, no test results
780	6.6.1.3	8	PASS
781	6.6.1.4	9	PASS, NO_OPTION
782	6.6.1.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
783	6.6.1.4	11	PASS, NO_OPTION
784	6.6.1.4	12	PASS, NO_OPTION, NO_TEST

785 **Assertions for fdatsync**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
786			
787	6.6.2.1	1	PASS[1,2], NO_OPTION
788	6.6.2.1	2	PASS[1,2], NO_OPTION
789	6.6.2.1	3	PASS, NO_OPTION
790	6.6.2.1	4	PASS, NO_OPTION
791	6.6.2.2	5	PASS, NO_OPTION, NO_TEST
792	6.6.2.2	R_1	Reference Assertion, no test results
793	6.6.2.2	6	PASS, NO_OPTION, NO_TEST
794	6.6.2.2	7	PASS, NO_OPTION, NO_TEST
795	6.6.2.3	8	PASS
796	6.6.2.4	9	PASS, NO_OPTION
797	6.6.2.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
798	6.6.2.4	11	PASS, NO_OPTION
799	6.6.2.4	12	PASS, NO_OPTION, NO_TEST

800 **Assertions for aio.h**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
801			
802	6.7.1.1	1	PASS
803	6.7.1.1	D_1	PASS
804	6.7.1.1	2	PASS, NO_TEST
805	6.7.1.1	3	PASS, NO_OPTION
806	6.7.1.1	4	PASS, NO_OPTION, NO_TEST_SUPPORT
807	6.7.1.1	D_2	PASS
808	6.7.1.1	5	PASS, NO_OPTION
809	6.7.1.1	6	PASS, NO_OPTION
810	6.7.1.1	7	PASS, NO_OPTION, NO_TEST_SUPPORT
811	6.7.1.1	8	PASS, NO_OPTION, NO_TEST_SUPPORT
812	6.7.1.1	D_3	PASS, NO_OPTION

813	6.7.1.1	9	PASS, NO_OPTION
814	6.7.1.1	10	PASS, NO_OPTION
815	6.7.1.1	D_4	PASS
816	6.7.1.1	11	PASS, NO_OPTION
817	6.7.1.1	12	PASS, NO_OPTION
818	6.7.1.1	13	PASS, NO_OPTION
819	6.7.1.1	14	PASS, NO_OPTION
820	6.7.1.2	15	PASS
821	6.7.1.2	16	PASS
822	6.7.1.2	17	PASS

---

823 **Assertions for aio\_read**

824	Subclause	Assertion ID	Conforming Results
825	6.7.2.1	1	PASS[1, 2], NO_OPTION
826	6.7.2.1	2	PASS[1, 2], NO_OPTION
827	6.7.2.1	3	PASS, NO_OPTION
828	6.7.2.1	4	PASS, NO_OPTION
829	6.7.2.2	5	PASS, NO_OPTION
830	6.7.2.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
831	6.7.2.2	7	PASS, NO_OPTION, NO_TEST
832	6.7.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
833	6.7.2.2	9	PASS, NO_OPTION, NO_TEST
834	6.7.2.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
835	6.7.2.2	11	PASS, NO_OPTION, NO_TEST
836	6.7.2.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT
837	6.7.2.2	R_1	Reference Assertion, no test results
838	6.7.2.2	13	PASS, NO_OPTION
839	6.7.2.2	D_1	PASS, NO_OPTION
840	6.7.2.2	14	PASS, NO_OPTION
841	6.7.2.2	D_2	PASS, NO_OPTION
842	6.7.2.2	D_3	PASS, NO_OPTION
843	6.7.2.2	15	PASS, NO_OPTION, NO_TEST
844	6.7.2.2	16	PASS, NO_OPTION, NO_TEST
845	6.7.2.2	17	PASS, NO_OPTION, NO_TEST
846	6.7.2.2	D_4	PASS, NO_OPTION
847	6.7.2.3	R_2	Reference Assertion, no test results
848	6.7.2.3	R_3	Reference Assertion, no test results
849	6.7.2.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
850	6.7.2.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
851	6.7.2.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT
852	6.7.2.4	21	PASS, NO_OPTION
853	6.7.2.4	ebadf1	PASS, NO_OPTION, NO_TEST_SUPPORT
854	6.7.2.4	ebadf2	PASS, NO_OPTION, NO_TEST_SUPPORT
855	6.7.2.4	EINVAL1	PASS, NO_OPTION, NO_TEST_SUPPORT
856	6.7.2.4	22	PASS, NO_OPTION, NO_TEST_SUPPORT
857	6.7.2.4	23	PASS, NO_OPTION, NO_TEST_SUPPORT
858	6.7.2.4	24	PASS, NO_OPTION, NO_TEST_SUPPORT, NO TEST

859	6.7.2.4	25	PASS, NO_OPTION, NO_TEST_SUPPORT
860	6.7.2.4	26	PASS, NO_OPTION, NO_TEST_SUPPORT
861	6.7.2.4	27	PASS, NO_OPTION, NO_TEST_SUPPORT
862	6.7.2.4	28	PASS, NO_OPTION, NO_TEST_SUPPORT
863	6.7.2.4	R_4	Reference Assertion, no test results
864	6.7.2.4	R_5	Reference Assertion, no test results
865	6.7.2.4	29	PASS, NO_OPTION, NO_TEST_SUPPORT
866	6.7.2.4	R_6	Reference Assertion, no test results

---

867 **Assertions for aio\_write**

868	Subclause	Assertion ID	Conforming Results
869	6.7.3.1	1	PASS[1, 2], NO_OPTION
870	6.7.3.1	2	PASS[1, 2], NO_OPTION
871	6.7.3.1	3	PASS, NO_OPTION
872	6.7.3.1	4	PASS, NO_OPTION
873	6.7.3.2	5	PASS, NO_OPTION
874	6.7.3.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
875	6.7.3.2	7	PASS, NO_OPTION, NO_TEST
876	6.7.3.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
877	6.7.3.2	9	PASS, NO_OPTION, NO_TEST
878	6.7.3.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
879	6.7.3.2	11	PASS, NO_OPTION, NO_TEST
880	6.7.3.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT
881	6.7.3.2	13	PASS, NO_OPTION
882	6.7.3.2	14	PASS, NO_OPTION
883	6.7.3.2	D_2	PASS, NO_OPTION
884	6.7.3.2	15	PASS, NO_OPTION
885	6.7.3.2	D_3	PASS, NO_OPTION
886	6.7.3.2	16	PASS, NO_OPTION, NO_TEST
887	6.7.3.2	17	PASS, NO_OPTION, NO_TEST
888	6.7.3.2	D_4	PASS, NO_OPTION
889	6.7.3.3	R_1	Reference Assertion, no test results
890	6.7.3.3	R_2	Reference Assertion, no test results
891	6.7.3.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
892	6.7.3.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
893	6.7.3.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT
894	6.7.3.4	21	PASS, NO_OPTION
895	6.7.3.4	ebadf1	PASS, NO_OPTION, NO_TEST_SUPPORT
896	6.7.3.4	ebadf2	PASS, NO_OPTION, NO_TEST_SUPPORT
897	6.7.3.4	EINVAL	PASS, NO_OPTION, NO_TEST_SUPPORT
898	6.7.3.4	22	PASS, NO_OPTION, NO_TEST_SUPPORT
899	6.7.3.4	23	PASS, NO_OPTION, NO_TEST_SUPPORT
900	6.7.3.4	24	PASS, NO_OPTION, NO_TEST_SUPPORT
901	6.7.3.4	25	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
902	6.7.3.4	26	PASS, NO_OPTION, NO_TEST_SUPPORT
903	6.7.3.4	27	PASS, NO_OPTION, NO_TEST_SUPPORT
904	6.7.3.4	28	PASS, NO_OPTION, NO_TEST_SUPPORT

905	6.7.3.4	R_3	Reference Assertion, no test results
906	6.7.3.4	R_4	Reference Assertion, no test results
907	6.7.3.4	29	PASS, NO_OPTION, NO_TEST_SUPPORT
908	6.7.3.4	R_5	Reference Assertion, no test results

---

909 **Assertions for lio\_listio**

910	Subclause	Assertion ID	Conforming Results
911	6.7.4.1	1	PASS[1, 2], NO_OPTION
912	6.7.4.1	2	PASS[1, 2], NO_OPTION
913	6.7.4.1	3	PASS, NO_OPTION
914	6.7.4.1	4	PASS, NO_OPTION
915	6.7.4.2	5	PASS, NO_OPTION
916	6.7.4.2	6	PASS, NO_OPTION
917	6.7.4.2	7	PASS, NO_OPTION
918	6.7.4.2	8	PASS, NO_OPTION
919	6.7.4.2	D_1	PASS, NO_OPTION
920	6.7.4.2	9	PASS, NO_OPTION
921	6.7.4.2	10	PASS, NO_OPTION
922	6.7.4.2	lio_read_op	PASS, NO_OPTION
923	6.7.4.2	lio_write_op	PASS, NO_OPTION
924	6.7.4.2	R_1	Reference Assertion, no test results
925	6.7.4.2	11	PASS, NO_OPTION, NO_TEST
926	6.7.4.2	12	PASS, NO_OPTION, NO_TEST
927	6.7.4.2	13	PASS, NO_OPTION, NO_TEST
928	6.7.4.2	14	PASS, NO_OPTION, NO_TEST
929	6.7.4.2	15	PASS, NO_OPTION, NO_TEST
930	6.7.4.3	R_2	Reference Assertion, no test results
931	6.7.4.3	R_3	Reference Assertion, no test results
932	6.7.4.3	R_4	Reference Assertion, no test results
933	6.7.4.3	R_5	Reference Assertion, no test results
934	6.7.4.3	16	PASS, NO_OPTION
935	6.7.4.3	R_6	Reference Assertion, no test results
936	6.7.4.4	17	PASS, NO_OPTION, NO_TEST
937	6.7.4.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
938	6.7.4.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
939	6.7.4.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT
940	6.7.4.4	lio_read_ebadf1	PASS, NO_OPTION, NO_TEST_SUPPORT
941	6.7.4.4	lio_read_ebadf2	PASS, NO_OPTION, NO_TEST_SUPPORT
942	6.7.4.4	lio_read_einval1	PASS, NO_OPTION, NO_TEST_SUPPORT
943	6.7.4.4	21	PASS, NO_OPTION, NO_TEST_SUPPORT
944	6.7.4.4	22	PASS, NO_OPTION, NO_TEST_SUPPORT
945	6.7.4.4	23	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
946	6.7.4.4	24	PASS, NO_OPTION, NO_TEST_SUPPORT
947	6.7.4.4	25	PASS, NO_OPTION, NO_TEST_SUPPORT
948	6.7.4.4	26	PASS, NO_OPTION, NO_TEST_SUPPORT
949	6.7.4.4	27	PASS, NO_OPTION, NO_TEST_SUPPORT
950	6.7.4.4	R_7	Reference Assertion, no test results

951	6.7.4.4	R_8	Reference Assertion, no test results
952	6.7.4.4	28	PASS, NO_OPTION, NO_TEST_SUPPORT
953	6.7.4.4	lio_write_ebadf1	PASS, NO_OPTION, NO_TEST_SUPPORT
954	6.7.4.4	lio_write_ebadf2	PASS, NO_OPTION, NO_TEST_SUPPORT
955	6.7.4.4	lio_write_einval1	PASS, NO_OPTION, NO_TEST_SUPPORT
956	6.7.4.4	29	PASS, NO_OPTION, NO_TEST_SUPPORT
957	6.7.4.4	30	PASS, NO_OPTION, NO_TEST_SUPPORT
958	6.7.4.4	31	PASS, NO_OPTION, NO_TEST_SUPPORT
959	6.7.4.4	32	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
960	6.7.4.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT
961	6.7.4.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
962	6.7.4.4	35	PASS, NO_OPTION, NO_TEST_SUPPORT
963	6.7.4.4	R_10	Reference Assertion, no test results
964	6.7.4.4	R_11	Reference Assertion, no test results
965	6.7.4.4	36	PASS, NO_OPTION, NO_TEST_SUPPORT
966	6.7.4.4	37	PASS, NO_OPTION
967	6.7.4.4	38	PASS, NO_OPTION
968	6.7.4.4	39	PASS, NO_OPTION
969	6.7.4.4	40	PASS, NO_OPTION

970 **Assertions for aio\_error**

971	Subclause	Assertion ID	Conforming Results
972	6.7.5.1	1	PASS[1, 2], NO_OPTION
973	6.7.5.1	2	PASS[1, 2], NO_OPTION
974	6.7.5.1	3	PASS, NO_OPTION
975	6.7.5.1	4	PASS, NO_OPTION
976	6.7.5.2	5	PASS, NO_OPTION
977	6.7.5.2	R_1	Reference Assertion, no test results
978	6.7.5.2	6	PASS, NO_OPTION
979	6.7.5.3	7	PASS, NO_OPTION
980	6.7.5.3	R_2	Reference Assertion, no test results
981	6.7.5.3	8	PASS, NO_OPTION
982	6.7.5.4	9	PASS, NO_OPTION
983	6.7.5.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
984	6.7.5.4	11	PASS, NO_OPTION

985 **Assertions for aio\_return**

986	Subclause	Assertion ID	Conforming Results
987	6.7.6.1	1	PASS[1, 2], NO_OPTION
988	6.7.6.1	2	PASS[1, 2], NO_OPTION
989	6.7.6.1	3	PASS, NO_OPTION
990	6.7.6.1	4	PASS, NO_OPTION
991	6.7.6.2	return_status	PASS, NO_OPTION
992	6.7.6.2	R_1	Reference Assertion, no test results
993	6.7.6.2	D_1	PASS, NO_OPTION
994	6.7.6.2	R_2	Reference Assertion, no test results
995	6.7.6.2	5	PASS, NO_OPTION



996	6.7.6.3	R_3	Reference Assertion, no test results
997	6.7.6.3	D_2	PASS, NO_OPTION
998	6.7.6.4	EINVAL	PASS, NO_OPTION
999	6.7.6.4	6	PASS, NO_OPTION
1000	6.7.6.4	7	PASS, NO_OPTION

1001 **Assertions for aio\_cancel**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1003	6.7.7.1	1	PASS[1, 2], NO_OPTION
1004	6.7.7.1	2	PASS[1, 2], NO_OPTION
1005	6.7.7.1	3	PASS, NO_OPTION
1006	6.7.7.1	4	PASS, NO_OPTION
1007	6.7.7.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
1008	6.7.7.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
1009	6.7.7.2	R_1	Reference Assertion, no test results
1010	6.7.7.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1011	6.7.7.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
1012	6.7.7.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
1013	6.7.7.2	D_1	PASS, NO_OPTION
1014	6.7.7.2	D_2	PASS, NO_OPTION
1015	6.7.7.3	R_2	Reference Assertion, no test results
1016	6.7.7.3	10	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1017	6.7.7.3	11	PASS, NO_OPTION, NO_TEST_SUPPORT
1018	6.7.7.3	R_3	Reference Assertion, no test results
1019	6.7.7.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
1020	6.7.7.4	13	PASS, NO_OPTION

1021 **Assertions for aio\_suspend**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1023	6.7.8.1	1	PASS[1, 2], NO_OPTION
1024	6.7.8.1	2	PASS[1, 2], NO_OPTION
1025	6.7.8.1	3	PASS, NO_OPTION
1026	6.7.8.1	4	PASS, NO_OPTION
1027	6.7.8.2	completion	PASS, NO_OPTION, NO_TEST_SUPPORT
1028	6.7.8.2	interrupt	PASS, NO_OPTION, NO_TEST_SUPPORT
1029	6.7.8.2	timeout	PASS, NO_OPTION, NO_TEST_SUPPORT
1030	6.7.8.2	already_completed	PASS, NO_OPTION, NO_TEST_SUPPORT
1031	6.7.8.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
1032	6.7.8.2	D_1	PASS, NO_OPTION
1033	6.7.8.2	R_1	Reference Assertion, no test results
1034	6.7.8.3	R_2	Reference Assertion, no test results
1035	6.7.8.3	R_3	Reference Assertion, no test results
1036	6.7.8.4	R_4	Reference Assertion, no test results
1037	6.7.8.4	6	PASS
1038	6.7.8.4	no_support	PASS, NO_OPTION

1039 **Assertions of aio\_fsync**

1040	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1041	6.7.9.1	1	PASS[1, 2], NO_OPTION
1042	6.7.9.1	2	PASS[1, 2], NO_OPTION
1043	6.7.9.1	3	PASS, NO_OPTION
1044	6.7.9.1	4	PASS, NO_OPTION
1045	6.7.9.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1046	6.7.9.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1047	6.7.9.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1048	6.7.9.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1049	6.7.9.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1050	6.7.9.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
1051	6.7.9.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
1052	6.7.9.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT
1053	6.7.9.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT
1054	6.7.9.2	D_1	PASS, NO_OPTION
1055	6.7.9.3	R_1	Reference Assertion, no test results
1056	6.7.9.3	R_2	Reference Assertion, no test results
1057	6.7.9.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
1058	6.7.9.4	15	PASS, NO_OPTION, NO_TEST_SUPPORT
1059	6.7.9.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
1060	6.7.9.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT
1061	6.7.9.4	18	PASS, NO_OPTION
1062	6.7.9.4	19	PASS, NO_OPTION

1063 **A.7 Device- and Class-Specific Functions**1064 **Assertions for Section\_7**

1065	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1066	No assertions for Section_7		

1067 **A.8 Language-Specific Services for the C Programming Language**

1068	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1069	8.2.2.2	D_1	PASS, NO_OPTION

1070 **A.9 System Databases**1071 **Assertions for Section\_9**

1072	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1073	No assertions for Section_9		

1074 **A.10 Data Interchange Format**1075 **Assertions for Section\_10**

1076	Subclause	Assertion ID	Conforming Results
1077	No assertions for Section_10		

1078 **A.11 Synchronization**1079 **Assertions for sem\_hdr**

1080	Subclause	Assertion ID	Conforming Results
1081	11.1	1	PASS
1082	11.1	GA_semOpenMaxFD	General Assertion, no test results
1083	11.1	GA_semPCTSOpenMaxFD	General Assertion, no test results

1084 **Assertions for sem\_init**

1085	Subclause	Assertion ID	Conforming Results
1086	11.2.1.1	1	PASS[1, 2], NO_OPTION
1087	11.2.1.1	2	PASS[1, 2], NO_OPTION
1088	11.2.1.1	3	PASS, NO_OPTION
1089	11.2.1.1	4	PASS, NO_OPTION
1090	11.2.1.2	OpenMaxSems	PASS[OpenMaxSems, PCTSOpenMaxSems]
1091	11.2.1.2	PCTS OpenMaxSems	PASS[OpenMaxSems, PCTSOpenMaxSems]
1092	11.2.1.2	sem_init	PASS, NO_OPTION, NO_TEST_SUPPORT
1093	11.2.1.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
1094	11.2.1.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
1095	11.2.1.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
1096	11.2.1.2	D_1	PASS, NO_OPTION
1097	11.2.1.2	D_2	PASS, NO_OPTION
1098	11.2.1.2	D_3	PASS, NO_OPTION
1099	11.2.1.3	R_1	Reference Assertion, no test results
1100	11.2.1.3	R_2	Reference Assertion, no test results
1101	11.2.1.4	8	PASS, NO_OPTION, NO_TEST_SUPPORT
1102	11.2.1.4	9	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1103	11.2.1.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
1104	11.2.1.4	11	PASS, NO_OPTION, NO_TEST_SUPPORT
1105	11.2.1.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
1106	11.2.1.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT
1107	11.2.1.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
1108	11.2.1.4	15	PASS, NO_OPTION, NO_TEST_SUPPORT

1109 **Assertions for sem-destroy**

1110	Subclause	Assertion ID	Conforming Results
1111	11.2.2.1	1	PASS[1, 2], NO_OPTION
1112	11.2.1.1	2	PASS[1, 2], NO_OPTION

1113	11.2.2.1	3	PASS, NO_OPTION
1114	11.2.2.1	4	PASS, NO_OPTION
1115	11.2.2.2	sem_destroy	PASS, NO_OPTION
1116	11.2.2.2	D_1	PASS, NO_OPTION
1117	11.2.2.2	D_2	PASS, NO_OPTION
1118	11.2.2.2	D_3	PASS, NO_OPTION
1119	11.2.2.2	D_4	PASS, NO_OPTION
1120	11.2.2.3	R_1	Reference Assertion, no test results
1121	11.2.2.3	R_2	Reference Assertion, no test results
1122	11.2.2.4	5	PASS, NO_OPTION
1123	11.2.2.4	6	PASS, NO_OPTION
1124	11.2.2.4	7	PASS, NO_OPTION

---

1125 **Assertions for sem\_open**

1126	Subclause	Assertion ID	Conforming Results
1127	11.2.3.1	1	PASS[1, 2], NO_OPTION
1128	11.2.3.1	2	PASS[1, 2], NO_OPTION
1129	11.2.3.1	3	PASS, NO_OPTION
1130	11.2.3.1	4	PASS, NO_OPTION
1131	11.2.3.2	OpenMaxSems	PASS[OpenMaxSems, PCTSOpenMaxSems]
1132	11.2.3.2	PCTSOpenMaxSems	PASS[OpenMaxSems, PCTSOpenMaxSems]
1133	11.2.3.2	sem_open	PASS, NO_OPTION, NO_TEST_SUPPORT
1134	11.2.3.2	5	PASS, NO_OPTION
1135	11.2.3.2	6	PASS, NO_OPTION
1136	11.2.3.2	7	PASS, NO_OPTION
1137	11.2.3.2	8	PASS, NO_OPTION
1138	11.2.3.2	9	PASS, NO_OPTION
1139	11.2.3.2	10	PASS, NO_OPTION
1140	11.2.3.2	11	PASS, NO_OPTION
1141	11.2.3.2	12	PASS, NO_OPTION
1142	11.2.3.2	D_1	PASS, NO_OPTION
1143	11.2.3.2	13	PASS, NO_OPTION
1144	11.2.3.2	14	PASS, NO_OPTION
1145	11.2.3.2	15	PASS, NO_OPTION, NO_TEST
1146	11.2.3.2	D_2	PASS, NO_OPTION
1147	11.2.3.2	D_3	PASS, NO_OPTION
1148	11.2.3.2	D_4	PASS, NO_OPTION
1149	11.2.3.2	D_5	PASS, NO_OPTION
1150	11.2.3.2	16	PASS, NO_OPTION
1151	11.2.3.2	17	PASS, NO_OPTION
1152	11.2.3.2	18	PASS, NO_OPTION
1153	11.2.3.2	19	PASS, NO_OPTION
1154	11.2.3.2	20	PASS, NO_OPTION
1155	11.2.3.2	D_6	PASS, NO_OPTION
1156	11.2.3.2	D_7	PASS, NO_OPTION
1157	11.2.3.2	21	PASS, NO_OPTION
1158	11.2.3.2	D_8	PASS, NO_OPTION

1159	11.2.3.2	D_9	PASS, NO_OPTION
1160	11.2.3.3	R_1	Reference Assertion, no test results
1161	11.2.3.4	22	PASS, NO_OPTION
1162	11.2.3.4	23	PASS, NO_OPTION
1163	11.2.3.4	24	PASS, NO_OPTION
1164	11.2.3.4	25	PASS, NO_OPTION
1165	11.2.3.4	D_10	PASS, NO_OPTION
1166	11.2.3.4	26	PASS, NO_OPTION
1167	11.2.3.4	27	PASS, NO_OPTION
1168	11.2.3.4	28	PASS, NO_OPTION, NO_TEST_SUPPORT
1169	11.2.3.4	29	PASS, NO_OPTION, NO_TEST_SUPPORT
1170	11.2.3.4	30	PASS, NO_OPTION, NO_TEST_SUPPORT
1171	11.2.3.4	31	PASS, NO_OPTION
1172	11.2.3.4	32	PASS, NO_OPTION
1173	11.2.3.4	33	PASS, NO_OPTION, NO_TEST
1174	11.2.3.4	34	PASS, NO_OPTION

1175 **Assertions for sem\_close**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1176			
1177	11.2.4.1	1	PASS[1, 2], NO_OPTION
1178	11.2.4.1	2	PASS[1, 2], NO_OPTION
1179	11.2.4.1	3	PASS, NO_OPTION
1180	11.2.4.1	4	PASS, NO_OPTION
1181	11.2.4.2	sem_close	PASS, NO_OPTION, NO_TEST_SUPPORT
1182	11.2.4.2	D_1	PASS, NO_OPTION
1183	11.2.4.2	D_2	PASS, NO_OPTION
1184	11.2.4.2	5	PASS, NO_OPTION
1185	11.2.4.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
1186	11.2.4.2	D_3	PASS, NO_OPTION
1187	11.2.4.3	R_1	Reference Assertion, no test results
1188	11.2.4.3	R_2	Reference Assertion, no test results
1189	11.2.4.4	7	PASS, NO_OPTION
1190	11.2.4.4	8	PASS, NO_OPTION

1191 **Assertions for sem\_unlink**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1191			
1192			
1193	11.2.5.1	1	PASS[1, 2], NO_OPTION
1194	11.2.5.1	2	PASS[1, 2], NO_OPTION
1195	11.2.5.1	3	PASS, NO_OPTION
1196	11.2.5.1	4	PASS, NO_OPTION
1197	11.2.5.2	sem_unlink	PASS, NO_OPTION
1198	11.2.5.2	5	PASS, NO_OPTION
1199	11.2.5.2	6	PASS, NO_OPTION
1200	11.2.5.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
1201	11.2.5.2	D_1	PASS, NO_OPTION
1202	11.2.5.3	R_1	Reference Assertion, no test results
1203	11.2.5.3	R_2	Reference Assertion, no test results

1204	11.2.5.4	9	PASS, NO_OPTION
1205	11.2.5.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
1206	11.2.5.4	11	PASS, NO_OPTION
1207	11.2.5.4	12	PASS, NO_OPTION

---

1208 **Assertions for sem\_wait**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1210	11.2.6.1	1	PASS[1, 2], NO_OPTION
1211	11.2.6.1	2	PASS[1, 2], NO_OPTION
1212	11.2.6.1	3	PASS, NO_OPTION
1213	11.2.6.1	4	PASS, NO_OPTION
1214	11.2.6.2	sem_wait	PASS, NO_OPTION, NO_TEST_SUPPORT
1215	11.2.6.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
1216	11.2.6.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT
1217	11.2.6.2	14	PASS, NO_OPTION, NO_TEST_SUPPORT
1218	11.2.6.2	D_1	PASS, NO_OPTION
1219	11.2.6.3	15	PASS, NO_OPTION, NO_TEST_SUPPORT
1220	11.2.6.3	R_1	Reference Assertion, no test results
1221	11.2.6.3	R_3	Reference Assertion, no test results
1222	11.2.6.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
1223	11.2.6.4	21	PASS, NO_OPTION, NO_TEST_SUPPORT
1224	11.2.6.4	23	PASS, NO_OPTION
1225	11.2.6.4	25	PASS, NO_OPTION, NO_TEST_SUPPORT

---

1226 **Assertions for sem\_trywait**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1228	11.2.6.1	5	PASS[5, 6], NO_OPTION
1229	11.2.6.1	6	PASS[5, 6], NO_OPTION
1230	11.2.6.1	7	PASS, NO_OPTION
1231	11.2.6.1	8	PASS, NO_OPTION
1232	11.2.6.2	sem_trywait	PASS, NO_OPTION, NO_TEST_SUPPORT
1233	11.2.6.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
1234	11.2.6.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
1235	11.2.6.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT
1236	11.2.6.2	D_2	PASS, NO_OPTION
1237	11.2.6.3	16	PASS, NO_OPTION, NO_TEST_SUPPORT
1238	11.2.6.3	R_2	Reference Assertion, no test results
1239	11.2.6.3	R_4	Reference Assertion, no test results
1240	11.2.6.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT
1241	11.2.6.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT
1242	11.2.6.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT
1243	11.2.6.4	22	PASS, NO_OPTION, NO_TEST_SUPPORT
1244	11.2.6.4	24	PASS, NO_OPTION
1245	11.2.6.4	26	PASS, NO_OPTION, NO_TEST_SUPPORT

---

1246 **Assertions for sem\_post**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1247			
1248	11.2.7.1	1	PASS[1, 2], NO_OPTION
1249	11.2.7.1	2	PASS[1, 2], NO_OPTION
1250	11.2.7.1	3	PASS, NO_OPTION
1251	11.2.7.1	4	PASS, NO_OPTION
1252	11.2.7.2	sem_post	PASS, NO_OPTION, NO_TEST_SUPPORT
1253	11.2.7.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
1254	11.2.7.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
1255	11.2.7.2	R_1	Reference Assertion, no test results
1256	11.2.7.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
1257	11.2.7.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
1258	11.2.7.2	D_1	PASS, NO_OPTION
1259	11.2.7.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
1260	11.2.7.2	D_2	PASS, NO_OPTION
1261	11.2.7.3	R_2	Reference Assertion, no test results
1262	11.2.7.3	R_3	Reference Assertion, no test results
1263	11.2.7.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
1264	11.2.7.4	11	PASS, NO_OPTION

1265 **Assertions for sem\_getvalue**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1266			
1267	11.2.8.1	1	PASS[1, 2], NO_OPTION
1268	11.2.8.1	2	PASS[1, 2], NO_OPTION
1269	11.2.8.1	3	PASS, NO_OPTION
1270	11.2.8.1	4	PASS, NO_OPTION
1271	11.2.8.2	sem_getvalue	PASS, NO_OPTION, NO_TEST_SUPPORT
1272	11.2.8.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
1273	11.2.8.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
1274	11.2.8.2	D_1	PASS, NO_OPTION
1275	11.2.8.3	R_1	Reference Assertion, no test results
1276	11.2.8.3	R_2	Reference Assertion, no test results
1277	11.2.8.4	7	PASS, NO_OPTION, NO_TEST_SUPPORT
1278	11.2.8.4	8	PASS, NO_OPTION

1279 **A.12 Memory Management**1280 **Assertions for mem-intro**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1281			
1282	12	1	PASS
1283	12	D_1	PASS, NO_OPTION
1284	12	D_2	PASS, NO_OPTION
1285	12	D_3	PASS, NO_OPTION
1286	12	R_1	Reference Assertion, no test results
1287	12	R_2	Reference Assertion, no test results
1288	12	R_3	Reference Assertion, no test results

1289	12	2	PASS, NO_TEST_SUPPORT
1290	12	3	PASS, NO_TEST_SUPPORT
1291	12	4	PASS, NO_TEST_SUPPORT
1292	12	R_4	Reference Assertion, no test results
1293	12	R_5	Reference Assertion, no test results
1294	12	D_4	PASS, NO_OPTION
1295	12	5	PASS, NO_OPTION, NO_TEST_SUPPORT
1296	12	6	PASS, NO_OPTION, NO_TEST_SUPPORT
1297	12	7	PASS, NO_OPTION, NO_TEST_SUPPORT
1298	12	R_6	Reference Assertion, no test results
1299	12	D_5	PASS, NO_OPTION

### 1300 Assertions for mlockall

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1301			
1302	12.1.1.1	1	PASS[1, 2], NO_OPTION
1303	12.1.1.1	2	PASS[1, 2], NO_OPTION
1304	12.1.1.1	3	PASS, NO_OPTION
1305	12.1.1.1	4	PASS, NO_OPTION
1306	12.1.1.2	mlockall	PASS, NO_OPTION, NO_TEST_SUPPORT
1307	12.1.1.2	9	PASS, NO_OPTION
1308	12.1.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
1309	12.1.1.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
1310	12.1.1.2	D_1	PASS, NO_OPTION
1311	12.1.1.2	14	PASS, NO_OPTION, NO_TEST_SUPPORT
1312	12.1.1.2	D_2	PASS, NO_OPTION
1313	12.1.1.2	15	PASS, NO_OPTION, NO_TEST_SUPPORT
1314	12.1.1.2	D_3	PASS, NO_OPTION
1315	12.1.1.2	D_4	PASS, NO_OPTION
1316	12.1.1.3	R_1	Reference Assertion, no test results
1317	12.1.1.3	R_2	Reference Assertion, no test results
1318	12.1.1.3	D_5	PASS, NO_OPTION
1319	12.1.1.4	16	PASS, NO_OPTION
1320	12.1.1.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT
1321	12.1.1.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
1322	12.1.1.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT
1323	12.1.1.4	21	PASS, NO_OPTION
1324	12.1.1.4	D_6	PASS, NO_OPTION
1325	12.1.1.4	22	PASS, NO_OPTION, NO_TEST_SUPPORT

### 1326 Assertions for munlockall

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1327			
1328	12.1.1.1	5	PASS[5, 6], NO_OPTION
1329	12.1.1.1	6	PASS[5, 6], NO_OPTION
1330	12.1.1.1	7	PASS, NO_OPTION
1331	12.1.1.1	8	PASS, NO_OPTION
1332	12.1.1.2	munlockall	PASS, NO_OPTION
1333	12.1.1.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT



1334	12.1.1.2	13	PASS, NO_OPTION
1335	12.1.1.3	R_3	Reference Assertion, no test results
1336	12.1.1.4	17	PASS, NO_OPTION

---

1337 **Assertions for mlock**

1338	Subclause	Assertion ID	Conforming Results
1339	12.1.2.1	1	PASS[1, 2], NO_OPTION
1340	12.1.2.1	2	PASS[1, 2], NO_OPTION
1341	12.1.2.1	3	PASS, NO_OPTION
1342	12.1.2.1	4	PASS, NO_OPTION
1343	12.1.2.2	mlock	PASS, NO_OPTION, NO_TEST_SUPPORT
1344	12.1.2.2	D_1	PASS, NO_OPTION
1345	12.1.2.2	D_3	PASS, NO_OPTION
1346	12.1.2.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
1347	12.1.2.2	D_4	PASS, NO_OPTION
1348	12.1.2.2	D_5	PASS, NO_OPTION
1349	12.1.2.3	R_1	Reference Assertion, no test results
1350	12.1.2.3	R_2	Reference Assertion, no test results
1351	12.1.2.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
1352	12.1.2.4	14	PASS, NO_OPTION
1353	12.1.2.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
1354	12.1.2.4	D_6	PASS, NO_OPTION
1355	12.1.2.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT
1356	12.1.2.4	D_7	PASS, NO_OPTION
1357	12.1.2.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
1358	12.1.2.4	D_8	PASS, NO_OPTION
1359	12.1.2.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT

---

1360 **Assertions for munlock**

1361	Subclause	Assertion ID	Conforming Results
1362	12.1.2.1	5	PASS[5, 6], NO_OPTION
1363	12.1.2.1	6	PASS[5, 6], NO_OPTION
1364	12.1.2.1	7	PASS, NO_OPTION
1365	12.1.2.1	8	PASS, NO_OPTION
1366	12.1.2.2	munlock	PASS, NO_OPTION, NO_TEST_SUPPORT
1367	12.1.2.2	D_2	PASS, NO_OPTION
1368	12.1.2.2	9	PASS, NO_OPTION
1369	12.1.2.2	10	PASS, NO_OPTION
1370	12.1.2.3	R_3	Reference Assertion, no test results
1371	12.1.2.4	13	PASS, NO_OPTION
1372	12.1.2.4	15	PASS, NO_OPTION
1373	12.1.2.4	18	PASS, NO_OPTION

---

1374 **Assertions for mmap**

1375	Subclause	Assertion ID	Conforming Results
1376	12.2.1.1	1	PASS[1, 2], NO_OPTION

1377	12.2.1.1	2	PASS[1, 2], NO_OPTION
1378	12.2.1.1	3	PASS, NO_OPTION
1379	12.2.1.1	4	PASS, NO_OPTION
1380	12.2.1.2	mmap	PASS, NO_OPTION
1381	12.2.1.2	D_1	PASS, NO_OPTION
1382	12.2.1.2	5	PASS, NO_OPTION
1383	12.2.1.2	6	PASS, NO_OPTION
1384	12.2.1.2	7	PASS, NO_OPTION
1385	12.2.1.2	8	PASS, NO_OPTION
1386	12.2.1.2	prot_values	PASS, NO_OPTION
1387	12.2.1.2	9	PASS, NO_OPTION
1388	12.2.1.2	R_1	Reference Assertion, no test results
1389	12.2.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
1390	12.2.1.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
1391	12.2.1.2	mem_protect_flags	PASS, NO_OPTION, NO_TEST_SUPPORT
1392	12.2.1.2	12	PASS, NO_OPTION
1393	12.2.1.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT
1394	12.2.1.2	D_2	PASS, NO_OPTION
1395	12.2.1.2	14	PASS, NO_OPTION, NO_TEST_SUPPORT
1396	12.2.1.2	15	PASS, NO_OPTION
1397	12.2.1.2	16	PASS, NO_OPTION, NO_TEST_SUPPORT
1398	12.2.1.2	D_3	PASS, NO_OPTION
1399	12.2.1.2	D_4	PASS, NO_OPTION
1400	12.2.1.2	17	PASS, NO_OPTION
1401	12.2.1.2	18	PASS, NO_OPTION
1402	12.2.1.2	19	PASS, NO_OPTION, NO_TEST_SUPPORT
1403	12.2.1.2	20	PASS, NO_OPTION, NO_TEST_SUPPORT
1404	12.2.1.2	D_5	PASS, NO_OPTION
1405	12.2.1.2	21	PASS, NO_OPTION, NO_TEST_SUPPORT
1406	12.2.1.2	D_6	PASS, NO_OPTION
1407	12.2.1.2	22	PASS, NO_OPTION, NO_TEST_SUPPORT
1408	12.2.1.2	23	PASS, NO_OPTION, NO_TEST_SUPPORT
1409	12.2.1.2	24	PASS, NO_OPTION
1410	12.2.1.2	25	PASS, NO_OPTION
1411	12.2.1.2	mmap_SIGBUS	PASS, NO_OPTION, NO_TEST_SUPPORT
1412	12.2.1.2	D_7	PASS, NO_OPTION
1413	12.2.1.2	D_8	PASS, NO_OPTION
1414	12.2.1.3	R_2	Reference Assertion, no test results
1415	12.2.1.3	R_3	Reference Assertion, no test results
1416	12.2.1.3	26	PASS, NO_OPTION
1417	12.2.1.3	R_4	Reference Assertion, no test results
1418	12.2.1.4	27	PASS, NO_OPTION
1419	12.2.1.4	28	PASS, NO_OPTION
1420	12.2.1.4	29	PASS, NO_OPTION, NO_TEST_SUPPORT
1421	12.2.1.4	30	PASS, NO_OPTION
1422	12.2.1.4	31	PASS, NO_OPTION, NO_TEST_SUPPORT
1423	12.2.1.4	32	PASS, NO_OPTION

1424	12.2.1.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1425	12.2.1.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
1426	12.2.1.4	35	PASS, NO_OPTION
1427	12.2.1.4	36	PASS, NO_OPTION, NO_TEST_SUPPORT
1428	12.2.1.4	37	PASS, NO_OPTION, NO_TEST_SUPPORT
1429	12.2.1.4	mmap_ENOTSUP	PASS, NO_OPTION
1430	12.2.1.4	38	PASS, NO_OPTION
1431	12.2.1.4	39	PASS, NO_OPTION, NO_TEST_SUPPORT
1432	12.2.1.4	40	PASS, NO_OPTION, NO_TEST_SUPPORT
1433	12.2.1.4	41	PASS, NO_OPTION, NO_TEST_SUPPORT

1434 **Assertions for munmap**

1435	Subclause	Assertion ID	Conforming Results
1436	12.2.2.1	1	PASS[1, 2], NO_OPTION
1437	12.2.2.1	2	PASS[1, 2], NO_OPTION
1438	12.2.2.1	3	PASS, NO_OPTION
1439	12.2.2.1	4	PASS, NO_OPTION
1440	12.2.2.2	munmap	PASS, NO_OPTION
1441	12.2.2.2	munmap_SIGSEGV	PASS, NO_OPTION
1442	12.2.2.2	5	PASS, NO_OPTION
1443	12.2.2.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
1444	12.2.2.2	D_1	PASS, NO_OPTION
1445	12.2.2.2	munlock_remove_maps	PASS, NO_OPTION, NO_TEST_SUPPORT
1446	12.2.2.2	D_2	PASS, NO_OPTION
1447	12.2.2.2	D_3	PASS, NO_OPTION
1448	12.2.2.3	R_1	Reference Assertion, no test results
1449	12.2.2.3	R_2	Reference Assertion, no test results
1450	12.2.2.4	8	PASS, NO_OPTION, NO_TEST
1451	12.2.2.4	9	PASS, NO_OPTION
1452	12.2.2.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT

1453 **Assertions for mprotect**

1454	Subclause	Assertion ID	Conforming Results
1455	12.2.3.1	1	PASS[1, 2], NO_OPTION
1456	12.2.3.1	2	PASS[1, 2], NO_OPTION
1457	12.2.3.1	3	PASS, NO_OPTION
1458	12.2.3.1	4	PASS, NO_OPTION
1459	12.2.3.2	mprotect	PASS, NO_OPTION
1460	12.2.3.2	R_1	Reference Assertion, no test results
1461	12.2.3.2	5	PASS, NO_OPTION
1462	12.2.3.2	R_2	Reference Assertion, no test results
1463	12.2.3.2	D_1	PASS, NO_OPTION
1464	12.2.3.2	6	PASS, NO_OPTION
1465	12.2.3.2	7	PASS, NO_OPTION
1466	12.2.3.2	8	PASS, NO_OPTION
1467	12.2.3.2	9	PASS, NO_OPTION

1468	12.2.3.2	D_2	PASS, NO_OPTION
1469	12.2.3.2	D_3	PASS, NO_OPTION
1470	12.2.3.2	D_4	PASS, NO_OPTION
1471	12.2.3.3	R_3	Reference Assertion, no test results
1472	12.2.3.3	R_4	Reference Assertion, no test results
1473	12.2.3.3	10	PASS, NO_OPTION
1474	12.2.3.4	11	PASS, NO_OPTION
1475	12.2.3.4	12	PASS, NO_OPTION
1476	12.2.3.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TESTS
1477	12.2.3.4	14	PASS, NO_OPTION
1478	12.2.3.4	15	PASS, NO_OPTION
1479	12.2.3.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
1480	12.2.3.4	17	PASS, NO_OPTION
1481	12.2.3.4	mprotect_ENOTSUP	PASS, NO_OPTION
1482	12.2.3.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT

---

1483 **Assertions for msync**

1484	Subclause	Assertion ID	Conforming Results
1485	12.2.4.1	1	PASS[1, 2], NO_OPTION
1486	12.2.4.1	2	PASS[1, 2], NO_OPTION
1487	12.2.4.1	3	PASS, NO_OPTION
1488	12.2.4.1	4	PASS, NO_OPTION
1489	12.2.4.2	msync	PASS, NO_OPTION, NO_TEST_SUPPORT
1490	12.2.4.2	5	PASS, NO_OPTION, NO_TEST
1491	12.2.4.2	D_1	PASS, NO_OPTION
1492	12.2.4.2	D_2	PASS, NO_OPTION
1493	12.2.4.2	D_3	PASS, NO_OPTION
1494	12.2.4.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
1495	12.2.4.2	D_4	PASS, NO_OPTION
1496	12.2.4.2	D_5	PASS, NO_OPTION
1497	12.2.4.2	7	PASS, NO_OPTION
1498	12.2.4.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1499	12.2.4.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1500	12.2.4.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1501	12.2.4.2	R_1	Reference Assertion, no test results
1502	12.2.4.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1503	12.2.4.2	D_6	PASS, NO_OPTION
1504	12.2.4.2	D_7	PASS, NO_OPTION
1505	12.2.4.3	R_2	Reference Assertion, no test results
1506	12.2.4.3	R_3	Reference Assertion, no test results
1507	12.2.4.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
1508	12.2.4.4	msync_einval	PASS, NO_OPTION, NO_TEST_SUPPORT
1509	12.2.4.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT
1510	12.2.4.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
1511	12.2.4.4	15	PASS, NO_OPTION
1512	12.2.4.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT

---

1513 **Assertions for shm\_open**

1514	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1515	12.3.1.1	1	PASS[1, 2], NO_OPTION
1516	12.3.1.1	2	PASS[1, 2], NO_OPTION
1517	12.3.1.1	3	PASS, NO_OPTION
1518	12.3.1.1	4	PASS, NO_OPTION
1519	12.3.1.2	shm_open	PASS, NO_OPTION
1520	12.3.1.2	5	PASS, NO_OPTION
1521	12.3.1.2	D_1	PASS, NO_OPTION
1522	12.3.1.2	6	PASS, NO_OPTION
1523	12.3.1.2	7	PASS, NO_OPTION
1524	12.3.1.2	8	PASS, NO_OPTION
1525	12.3.1.2	9	PASS, NO_OPTION
1526	12.3.1.2	10	PASS, NO_OPTION
1527	12.3.1.2	D_2	PASS, NO_OPTION
1528	12.3.1.2	D_3	PASS, NO_OPTION
1529	12.3.1.2	11	PASS, NO_OPTION
1530	12.3.1.2	12	PASS, NO_OPTION
1531	12.3.1.2	D_4	PASS, NO_OPTION
1532	12.3.1.2	13	PASS, NO_OPTION
1533	12.3.1.2	14	PASS, NO_OPTION
1534	12.3.1.2	15	PASS, NO_OPTION
1535	12.3.1.2	16	PASS, NO_OPTION
1536	12.3.1.2	17	PASS, NO_OPTION
1537	12.3.1.2	18	PASS, NO_OPTION
1538	12.3.1.2	19	PASS, NO_OPTION
1539	12.3.1.2	20	PASS, NO_OPTION
1540	12.3.1.2	21	PASS, NO_OPTION
1541	12.3.1.2	22	PASS, NO_OPTION
1542	12.3.1.2	23	PASS, NO_OPTION
1543	12.3.1.2	24	PASS, NO_OPTION
1544	12.3.1.2	D_5	PASS, NO_OPTION
1545	12.3.1.2	25	PASS, NO_OPTION
1546	12.3.1.2	26	PASS, NO_OPTION
1547	12.3.1.2	R_1	Reference Assertion, no test results
1548	12.3.1.2	27	PASS, NO_OPTION, NO_TEST
1549	12.3.1.2	D_6	PASS, NO_OPTION
1550	12.3.1.2	28	PASS, NO_OPTION
1551	12.3.1.2	29	PASS, NO_OPTION
1552	12.3.1.2	D_7	PASS, NO_OPTION
1553	12.3.1.2	30	PASS, NO_OPTION
1554	12.3.1.2	D_8	PASS, NO_OPTION
1555	12.3.1.2	D_9	PASS, NO_OPTION
1556	12.3.1.3	R_2	Reference Assertion, no test results
1557	12.3.1.3	R_3	Reference Assertion, no test results
1558	12.3.1.4	31	PASS, NO_OPTION
1559	12.3.1.4	32	PASS, NO_OPTION

1560	12.3.1.4	33	PASS, NO_OPTION
1561	12.3.1.4	34	PASS, NO_OPTION
1562	12.3.1.4	35	PASS, NO_OPTION
1563	12.3.1.4	D_10	PASS, NO_OPTION
1564	12.3.1.4	36	PASS, NO_OPTION
1565	12.3.1.4	37	PASS, NO_OPTION, NO_TEST_SUPPORT
1566	12.3.1.4	38	PASS, NO_OPTION, NO_TEST_SUPPORT
1567	12.3.1.4	39	PASS, NO_OPTION
1568	12.3.1.4	40	PASS, NO_OPTION
1569	12.3.1.4	41	PASS, NO_OPTION, NO_TEST
1570	12.3.1.4	42	PASS, NO_OPTION

#### 1571 **Assertions for shm\_unlink**

1572	Subclause	Assertion ID	Conforming Results
1573	12.3.2.1	1	PASS[1, 2], NO_OPTION
1574	12.3.2.1	2	PASS[1, 2], NO_OPTION
1575	12.3.2.1	3	PASS, NO_OPTION
1576	12.3.2.1	4	PASS, NO_OPTION
1577	12.3.2.2	shm_unlink	PASS, NO_OPTION
1578	12.3.2.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
1579	12.3.2.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
1580	12.3.2.2	D_1	PASS, NO_OPTION
1581	12.3.2.3	R_1	Reference Assertion, no test results
1582	12.3.2.3	R_2	Reference Assertion, no test results
1583	12.3.2.4	7	PASS, NO_OPTION
1584	12.3.2.4	8	PASS, NO_OPTION, NO_TEST_RESULTS
1585	12.3.2.4	9	PASS, NO_OPTION
1586	12.3.2.4	10	PASS, NO_OPTION

### 1587 **A.13 Execution Scheduling**

#### 1588 **Assertions for sched\_param**

1589	Subclause	Assertion ID	Conforming Results
1590	13.1	1	PASS
1591	13.1	D_1	PASS, NO_OPTION
1592	13.1	2	PASS, NO_TEST
1593	13.1	3	PASS

#### 1594 **Assertions for sched\_policy**

1595	Subclause	Assertion ID	Conforming Results
1596	13.2	1	PASS
1597	13.2	2	PASS, NO_TEST
1598	13.2	D_1	PASS, NO_OPTION
1599	13.2	3	PASS
1600	13.2.1	sched_fifo1	PASS, NO_OPTION, NO_TEST_SUPPORT

1601	13.2.1	sched_fifo2	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1602	13.2.1	sched_fifo3	PASS, NO_OPTION, NO_TEST_SUPPORT
1603	13.2.1	sched_fifo4	PASS, NO_OPTION, NO_TEST_SUPPORT
1604	13.2.1	sched_fifo5	PASS, NO_OPTION, NO_TEST_SUPPORT
1605	13.2.1	sched_fifo6	PASS, NO_OPTION, NO_TEST_SUPPORT
1606	13.2.1	sched_fifo7	PASS, NO_OPTION, NO_TEST_SUPPORT
1607	13.2.1	sched_fifo8	PASS, NO_OPTION, NO_TEST_SUPPORT
1608	13.2.1	R_1	Reference Assertion, no test results
1609	13.2.1	4	PASS, NO_OPTION, NO_TEST_SUPPORT
1610	13.2.1	5	PASS, NO_OPTION, NO_TEST_SUPPORT
1611	13.2.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
1612	13.2.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1613	13.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
1614	13.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
1615	13.2.3	D_2	PASS
1616	13.2.3	D_3	PASS
1617	13.2.3	10	PASS, NO_OPTION, NO_TEST_SUPPORT

1618 **Assertions for sched\_setparam**

1619	Subclause	Assertion ID	Conforming Results
1620	13.3.1.1	1	PASS[1, 2], NO_OPTION
1621	13.3.1.1	2	PASS[1, 2], NO_OPTION
1622	13.3.1.1	3	PASS, NO_OPTION
1623	13.3.1.1	4	PASS, NO_OPTION
1624	13.3.1.2	sched_setparam	PASS, NO_OPTION, NO_TEST_SUPPORT
1625	13.3.1.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
1626	13.3.1.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1627	13.3.1.2	D_1	PASS, NO_OPTION
1628	13.3.1.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
1629	13.3.1.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
1630	13.3.1.2	D_2	PASS, NO_OPTION
1631	13.3.1.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
1632	13.3.1.2	D_3	PASS, NO_OPTION
1633	13.3.1.2	10	PASS, NO_OPTION, NO_TEST
1634	13.3.1.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1635	13.3.1.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1636	13.3.1.2	D_4	PASS, NO_OPTION, NO_TEST_SUPPORT
1637	13.3.1.2	D_5	PASS, NO_OPTION
1638	13.3.1.3	R_1	Reference Assertion, no test results
1639	13.3.1.3	R_2	Reference Assertion, no test results
1640	13.3.1.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT
1641	13.3.1.4	14	PASS, NO_OPTION
1642	13.3.1.4	15	PASS, NO_OPTION, NO_TEST_SUPPORT
1643	13.3.1.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
1644	13.3.1.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT

1645 **Assertions for sched\_getparam**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1646			
1647	13.3.2.1	1	PASS[1, 2], NO_OPTION
1648	13.3.2.1	2	PASS[1, 2], NO_OPTION
1649	13.3.2.1	3	PASS, NO_OPTION
1650	13.3.2.1	4	PASS, NO_OPTION
1651	13.3.2.2	sched_getparam	PASS, NO_OPTION
1652	13.3.2.2	5	PASS, NO_OPTION
1653	13.3.2.2	6	PASS, NO_OPTION
1654	13.3.2.2	D_1	PASS, NO_OPTION
1655	13.3.2.2	D_2	PASS, NO_OPTION
1656	13.3.2.3	R_1	Reference Assertion, no test results
1657	13.3.2.3	R_2	Reference Assertion, no test results
1658	13.3.2.4	7	PASS, NO_OPTION
1659	13.3.2.4	8	PASS, NO_OPTION
1660	13.3.2.4	9	PASS, NO_OPTION

1661 **Assertions for sched\_setscheduler**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1662			
1663	13.3.3.1	1	PASS[1, 2], NO_OPTION
1664	13.3.3.1	2	PASS[1, 2], NO_OPTION
1665	13.3.3.1	3	PASS, NO_OPTION
1666	13.3.3.1	4	PASS, NO_OPTION
1667	13.3.3.2	sched_setscheduler	PASS, NO_OPTION, NO_TEST_SUPPORT
1668	13.3.3.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
1669	13.3.3.2	D_1	PASS, NO_OPTION
1670	13.3.3.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
1671	13.3.3.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
1672	13.3.3.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
1673	13.3.3.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
1674	13.3.3.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
1675	13.3.3.2	D_2	PASS, NO_OPTION
1676	13.3.3.2	D_3	PASS, NO_OPTION, NO_TEST_SUPPORT
1677	13.3.3.2	D_4	PASS, NO_OPTION
1678	13.3.3.2	R_1	Reference Assertion, no test results
1679	13.3.3.2	D_5	PASS, NO_OPTION
1680	13.3.3.3	R_2	Reference Assertion, no test results
1681	13.3.3.3	R_3	Reference Assertion, no test results
1682	13.3.3.4	11	PASS, NO_OPTION, NO_TEST_SUPPORT
1683	13.3.3.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
1684	13.3.3.4	13	PASS, NO_OPTION
1685	13.3.3.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
1686	13.3.3.4	15	PASS, NO_OPTION, NO_TEST_SUPPORT
1687	13.3.3.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT

1688 **Assertions for sched\_getscheduler**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1689			
1690	13.3.4.1	1	PASS[1, 2], NO_OPTION



1691	13.3.4.1	2	PASS[1, 2], NO_OPTION
1692	13.3.4.1	3	PASS, NO_OPTION
1693	13.3.4.1	4	PASS, NO_OPTION
1694	13.3.4.2	sched_getscheduler	PASS, NO_OPTION
1695	13.3.4.2	5	PASS, NO_OPTION
1696	13.3.4.2	6	PASS, NO_OPTION
1697	13.3.4.2	7	PASS, NO_OPTION
1698	13.3.4.2	8	PASS, NO_OPTION
1699	13.3.4.2	D_1	PASS, NO_OPTION
1700	13.3.4.3	R_1	Reference Assertion, no test results
1701	13.3.4.3	R_2	Reference Assertion, no test results
1702	13.3.4.4	9	PASS, NO_OPTION
1703	13.3.4.4	10	PASS, NO_OPTION
1704	13.3.4.4	11	PASS, NO_OPTION

---

1705 **Assertions for sched\_yield**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1706			
1707	13.3.5.1	1	PASS[1, 2], NO_OPTION
1708	13.3.5.1	2	PASS[1, 2], NO_OPTION
1709	13.3.5.1	3	PASS, NO_OPTION
1710	13.3.5.1	4	PASS, NO_OPTION
1711	13.3.5.2	sched_yield	PASS, NO_OPTION, NO_TEST
1712	13.3.5.2	D_1	PASS, NO_OPTION
1713	13.3.5.3	R_1	Reference Assertion, no test results
1714	13.3.5.3	R_2	Reference Assertion, no test results
1715	13.3.5.4	5	PASS, NO_OPTION

1716 **Assertions for sched\_get\_priority\_max**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1717			
1718	13.3.6.1	1	PASS[1, 2], NO_OPTION
1719	13.3.6.1	2	PASS[1, 2], NO_OPTION
1720	13.3.6.1	3	PASS, NO_OPTION
1721	13.3.6.1	4	PASS, NO_OPTION
1722	13.3.6.2	sched_get_priority_max	PASS, NO_OPTION
1723	13.3.6.2	14	PASS, NO_OPTION
1724	13.3.6.2	D_1	PASS, NO_OPTION
1725	13.3.6.3	R_1	Reference Assertion, no test results
1726	13.3.6.3	R_3	Reference Assertion, no test results
1727	13.3.6.4	16	PASS, NO_OPTION
1728	13.3.6.4	18	PASS, NO_OPTION
1729	13.3.6.4	19	PASS, NO_OPTION
1730	13.3.6.4	21	PASS, NO_OPTION

---

1731 **Assertions for sched\_get\_priority\_min**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1732			
1733	13.3.6.1	5	PASS[5, 6], NO_OPTION

1734	13.3.6.1	6	PASS[5, 6], NO_OPTION
1735	13.3.6.1	7	PASS, NO_OPTION
1736	13.3.6.1	8	PASS, NO_OPTION
1737	13.3.6.2	sched_get_priority_min	PASS, NO_OPTION
1738	13.3.6.2	15	PASS, NO_OPTION
1739	13.3.6.2	D_2	PASS, NO_OPTION
1740	13.3.6.3	R_2	Reference Assertion, no test results
1741	13.3.6.3	R_4	Reference Assertion, no test results
1742	13.3.6.4	17	PASS, NO_OPTION
1743	13.3.6.4	22	PASS, NO_OPTION

1744 **Assertions for sched\_rr\_get\_interval**

1745	Subclause	Assertion ID	Conforming Results
1746	13.3.6.1	9	PASS[9, 10], NO_OPTION
1747	13.3.6.1	10	PASS[9, 10], NO_OPTION
1748	13.3.6.1	11	PASS, NO_OPTION
1749	13.3.6.1	12	PASS, NO_OPTION
1750	13.3.6.2	sched_rr_get_interval	PASS, NO_OPTION, NO_TEST
1751	13.3.6.2	13	PASS, NO_OPTION, NO_TEST
1752	13.3.6.2	D_3	PASS, NO_OPTION
1753	13.3.6.2	R_5	Reference Assertion, no test results
1754	13.3.6.3	R_6	Reference Assertion, no test results
1755	13.3.6.4	20	PASS, NO_OPTION
1756	13.3.6.4	23	PASS, NO_OPTION

1757 **A.14 Clocks and Timers**

1758 **Assertions for timer\_hdr**

1759	Subclause	Assertion ID	Conforming Results
1760	14.1.1	1	PASS
1761	14.1.1	D_1	PASS, NO_OPTION
1762	14.1.1	2	PASS, NO_TEST
1763	14.1.1	3	PASS
1764	14.1.1	4	PASS, NO_TEST_SUPPORT
1765	14.1.1	5	PASS
1766	14.1.1	6	PASS
1767	14.1.1	D_2	PASS, NO_OPTION
1768	14.1.1	7	PASS, NO_TEST
1769	14.1.1	8	PASS
1770	14.1.1	9	PASS
1771	14.1.1	10	PASS
1772	14.1.1	11	PASS
1773	14.1.2	12	PASS, NO_TEST_SUPPORT
1774	14.1.3	13	PASS
1775	14.1.4	14	PASS
1776	14.1.4	15	PASS

1777	14.1.4	16	PASS
1778	14.1.4	D_3	PASS, NO_OPTION
1779	14.1.4	17	PASS
1780	14.1.4	D_4	PASS, NO_OPTION
1781	14.1.4	D_5	PASS
1782	14.1.4	18	PASS
1783	14.1.4	D_6	PASS

---

1784 **Assertions for clock-settime**

1785	Subclause	Assertion ID	Conforming Results
1786	14.2.1.1	1	PASS[1, 2], NO_OPTION
1787	14.2.1.1	2	PASS[1, 2], NO_OPTION
1788	14.2.1.1	3	PASS, NO_OPTION
1789	14.2.1.1	4	PASS, NO_OPTION
1790	14.2.1.2	clock_settime	PASS, NO_OPTION, NO_TEST_SUPPORT
1791	14.2.1.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT
1792	14.2.1.2	15	PASS
1793	14.2.1.2	17	PASS, NO_OPTION, NO_TEST_SUPPORT
1794	14.2.1.2	D_1	PASS
1795	14.2.1.2	18	PASS
1796	14.2.1.2	19	PASS
1797	14.2.1.2	20	PASS
1798	14.2.1.2	21	PASS
1799	14.2.1.2	23	PASS, NO_OPTION, NO_TEST_SUPPORT
1800	14.2.1.2	D_2	PASS, NO_OPTION
1801	14.2.1.2	24	PASS, NO_OPTION
1802	14.2.1.2	25	PASS, NO_OPTION
1803	14.2.1.2	D_3	PASS, NO_OPTION
1804	14.2.1.3	R_1	Reference Assertion, no test results
1805	14.2.1.3	R_2	Reference Assertion, no test results
1806	14.2.1.4	26	PASS, NO_OPTION, NO_TEST_SUPPORT
1807	14.2.1.4	29	PASS, NO_OPTION
1808	14.2.1.4	32	PASS, NO_OPTION, NO_TEST_SUPPORT
1809	14.2.1.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT
1810	14.2.1.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
1811	14.2.1.4	35	PASS, NO_OPTION, NO_TEST_SUPPORT

---

1812 **Assertions for clock\_gettime**

1813	Subclause	Assertion ID	Conforming Results
1814	14.2.1.1	5	PASS[5, 6], NO_OPTION
1815	14.2.1.1	6	PASS[5, 6], NO_OPTION
1816	14.2.1.1	7	PASS, NO_OPTION
1817	14.2.1.1	8	PASS, NO_OPTION
1818	14.2.1.2	clock_gettime	PASS, NO_OPTION
1819	14.2.1.2	22	PASS, NO_OPTION
1820	14.2.1.2	D_4	PASS, NO_OPTION
1821	14.2.1.3	R_3	Reference Assertion, no test results

1822	14.2.1.3	R_4	Reference Assertion, no test results
1823	14.2.1.4	27	PASS, NO_OPTION
1824	14.2.1.4	30	PASS, NO_OPTION

---

1825 **Assertions for clock\_getres**

1826	Subclause	Assertion ID	Conforming Results
1827	14.2.1.1	9	PASS[9, 10], NO_OPTION
1828	14.2.1.1	10	PASS[9, 10], NO_OPTION
1829	14.2.1.1	11	PASS, NO_OPTION
1830	14.2.1.1	12	PASS, NO_OPTION
1831	14.2.1.1	clock_getres	PASS, NO_OPTION
1832	14.2.1.2	14	PASS, NO_OPTION
1833	14.2.1.2	16	PASS, NO_OPTION
1834	14.2.1.2	D_5	PASS, NO_OPTION
1835	14.2.1.3	R_5	Reference Assertion, no test results
1836	14.2.1.3	R_6	Reference Assertion, no test results
1837	14.2.1.4	28	PASS, NO_OPTION
1838	14.2.1.4	31	PASS, NO_OPTION

---

1839 **Assertions for timer\_create**

1840	Subclause	Assertion ID	Conforming Results
1841	14.2.2.1	1	PASS[1, 2], NO_OPTION
1842	14.2.2.1	2	PASS[1, 2], NO_OPTION
1843	14.2.2.1	3	PASS, NO_OPTION
1844	14.2.2.1	4	PASS, NO_OPTION
1845	14.2.2.2	timer_create	PASS, NO_OPTION
1846	14.2.2.2	5	PASS, NO_OPTION
1847	14.2.2.2	6	PASS
1848	14.2.2.2	7	PASS, NO_OPTION
1849	14.2.2.2	8	PASS, NO_OPTION
1850	14.2.2.2	9	PASS, NO_OPTION
1851	14.2.2.2	10	PASS, NO_OPTION
1852	14.2.2.2	11	PASS, NO_OPTION
1853	14.2.2.2	12	PASS, NO_OPTION
1854	14.2.2.2	13	PASS, NO_OPTION
1855	14.2.2.2	14	PASS, NO_OPTION
1856	14.2.2.2	15	PASS, NO_OPTION
1857	14.2.2.2	D_1	PASS, NO_OPTION
1858	14.2.2.2	16	PASS, NO_OPTION
1859	14.2.2.2	17	PASS, NO_OPTION
1860	14.2.2.2	18	PASS, NO_OPTION, NO_TEST_SUPPORT
1861	14.2.2.2	19	PASS, NO_OPTION, NO_TEST_SUPPORT
1862	14.2.2.2	D_2	PASS, NO_OPTION
1863	14.2.2.2	20	PASS, NO_OPTION, NO_TEST_SUPPORT
1864	14.2.2.2	21	PASS, NO_OPTION, NO_TEST_SUPPORT
1865	14.2.2.2	22	PASS, NO_OPTION
1866	14.2.2.2	23	PASS, NO_OPTION

1867	14.2.2.2	24	PASS, NO_OPTION
1868	14.2.2.2	25	PASS, NO_OPTION
1869	14.2.2.3	R_1	Reference Assertion, no test results
1870	14.2.2.3	R_2	Reference Assertion, no test results
1871	14.2.2.3	D_4	PASS, NO_OPTION
1872	14.2.2.4	26	PASS, NO_OPTION, NO_TEST
1873	14.2.2.4	27	PASS, NO_OPTION, NO_TEST_SUPPORT
1874	14.2.2.4	28	PASS, NO_OPTION
1875	14.2.2.4	29	PASS, NO_OPTION

---

1876 **Assertions for timer\_delete**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1878	14.2.3.1	1	PASS[1, 2], NO_OPTION
1879	14.2.3.1	2	PASS[1, 2], NO_OPTION
1880	14.2.3.1	3	PASS, NO_OPTION
1881	14.2.3.1	4	PASS, NO_OPTION
1882	14.2.3.2	timer_delete	PASS, NO_OPTION, NO_TEST_SUPPORT
1883	14.2.3.2	5	PASS, NO_OPTION
1884	14.2.3.2	D_1	PASS, NO_OPTION
1885	14.2.3.2	D_2	PASS, NO_OPTION
1886	14.2.3.3	R_1	Reference Assertion, no test results
1887	14.2.3.3	R_2	Reference Assertion, no test results
1888	14.2.3.4	6	PASS, NO_OPTION
1889	14.2.3.4	7	PASS, NO_OPTION

---

1890 **Assertions for timer\_settime**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1892	14.2.4.1	1	PASS[1, 2], NO_OPTION
1893	14.2.4.1	2	PASS[1, 2], NO_OPTION
1894	14.2.4.1	3	PASS, NO_OPTION
1895	14.2.4.1	4	PASS, NO_OPTION
1896	14.2.4.2	timer_settime	PASS, NO_OPTION
1897	14.2.4.2	13	PASS, NO_OPTION
1898	14.2.4.2	14	PASS, NO_OPTION
1899	14.2.4.2	D_1	PASS, NO_OPTION
1900	14.2.4.2	15	PASS, NO_OPTION
1901	14.2.4.2	16	PASS, NO_OPTION
1902	14.2.4.2	17	PASS, NO_OPTION
1903	14.2.4.2	18	PASS, NO_OPTION
1904	14.2.4.2	19	PASS, NO_OPTION
1905	14.2.4.2	20	PASS, NO_OPTION
1906	14.2.4.2	21	PASS, NO_OPTION
1907	14.2.4.2	22	PASS, NO_OPTION
1908	14.2.4.2	23	PASS, NO_OPTION
1909	14.2.4.2	24	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST

1910	14.2.4.2	26	PASS, NO_TEST_SUPPORT
1911	14.2.4.2	27	PASS, NO_TEST_SUPPORT
1912	14.2.4.2	D_3	PASS, NO_OPTION
1913	14.2.4.3	R_1	Reference Assertion, no test results
1914	14.2.4.3	R_3	Reference Assertion, no test results
1915	14.2.4.4	32	PASS, NO_OPTION, NO_TEST_SUPPORT
1916	14.2.4.4	35	PASS, NO_OPTION
1917	14.2.4.4	38	PASS, NO_OPTION
1918	14.2.4.4	39	PASS, NO_OPTION, NO_TEST_SUPPORT

1919 **Assertions for timer\_gettime**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1921	14.2.4.1	5	PASS[5, 6], NO_OPTION
1922	14.2.4.1	6	PASS[5, 6], NO_OPTION
1923	14.2.4.1	7	PASS, NO_OPTION
1924	14.2.4.1	8	PASS, NO_OPTION
1925	14.2.4.2	timer_gettime	PASS, NO_OPTION
1926	14.2.4.2	25	PASS, NO_OPTION
1927	14.2.4.2	D_4	PASS, NO_OPTION
1928	14.2.4.3	R_2	Reference Assertion, no test results
1929	14.2.4.3	R_4	Reference Assertion, no test results
1930	14.2.4.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT
1931	14.2.4.4	36	PASS, NO_OPTION

1932 **Assertions for timer\_getoverrun**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1934	14.2.4.1	9	PASS[9, 10], NO_OPTION
1935	14.2.4.1	10	PASS[9, 10], NO_OPTION
1936	14.2.4.1	11	PASS, NO_OPTION
1937	14.2.4.1	12	PASS, NO_OPTION
1938	14.2.4.2	timer_getoverrun	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1939	14.2.4.2	28	PASS, NO_OPTION
1940	14.2.4.2	29	PASS, NO_OPTION, NO_TEST
1941	14.2.4.2	30	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
1942	14.2.4.2	31	PASS, NO_OPTION, NO_TEST
1943	14.2.4.2	D_2	PASS, NO_OPTION
1944	14.2.4.2	D_5	PASS, NO_OPTION
1945	14.2.4.3	R_5	Reference Assertion, no test results
1946	14.2.4.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
1947	14.2.4.4	37	PASS, NO_OPTION

1948 **Assertions for nanosleep**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1949	14.2.5.1	1	PASS[1, 2], NO_OPTION
1951	14.2.5.1	2	PASS[1, 2], NO_OPTION
1952	14.2.5.1	3	PASS, NO_OPTION

1953	14.2.5.1	4	PASS, NO_OPTION
1954	14.2.5.2	nanosleep	PASS, NO_OPTION
1955	14.2.5.2	5	PASS, NO_OPTION
1956	14.2.5.2	6	PASS, NO_OPTION, NO_TEST
1957	14.2.5.2	7	PASS, NO_OPTION, NO_TEST
1958	14.2.5.2	8	PASS, NO_OPTION
1959	14.2.5.2	9	PASS, NO_OPTION
1960	14.2.5.2	D_1	PASS, NO_OPTION
1961	14.2.5.3	R_1	Reference Assertion, no test results
1962	14.2.5.3	R_2	Reference Assertion, no test results
1963	14.2.5.3	10	PASS, NO_OPTION
1964	14.2.5.3	11	PASS, NO_OPTION
1965	14.2.5.3	R_3	Reference Assertion, no test results
1966	14.2.5.4	12	PASS, NO_OPTION
1967	14.2.5.4	13	PASS, NO_OPTION
1968	14.2.5.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
1969	14.2.5.4	15	PASS, NO_OPTION

---

1970 **A.15 Message Passing**

1971 **Assertions for mq\_intro**

1972	Subclause	Assertion ID	Conforming Results
1973	15.1	D_1	PASS, NO_OPTION

---

1974 **Assertions for mq\_hdr**

1975	Subclause	Assertion ID	Conforming Results
1976	15.1.1	1	PASS
1977	15.1.1	D_1	PASS
1978	15.1.1	2	PASS
1979	15.1.1	3	PASS
1980	15.1.1	D_2	PASS, NO_OPTION
1981	15.1.1	4	PASS, NO_TEST
1982	15.1.1	D_3	PASS, NO_OPTION
1983	15.1.1	GA_mqOpenMaxFD	General Assertion, no test results
1984	15.1.1	GA_mqPCTSOpenMaxFD	General Assertion, no test results

---

1985 **Assertions for mq\_open**

1986	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
1987	15.2.1.1	1	PASS[1, 2], NO_OPTION
1988	15.2.1.1	2	PASS[1, 2], NO_OPTION
1989	15.2.1.1	3	PASS, NO_OPTION
1990	15.2.1.1	4	PASS, NO_OPTION
1991	15.2.1.2	OpenMaxMqs	PASS[OpenMaxMqs, PCTSOpenMaxMqs]
1992	15.2.1.2	PCTSOpenMaxMqs	PASS[OpenMaxMqs, PCTSOpenMaxMqs]
1993	15.2.1.2	mq_open	PASS, NO_OPTION, NO_TEST_SUPPORT
1994	15.2.1.2	D_1	PASS, NO_OPTION, NO_TEST_SUPPORT
1995	15.2.1.2	5	PASS, NO_OPTION
1996	15.2.1.2	6	PASS, NO_OPTION
1997	15.2.1.2	7	PASS, NO_OPTION
1998	15.2.1.2	8	PASS, NO_OPTION
1999	15.2.1.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
2000	15.2.1.2	D_2	PASS, NO_OPTION
2001	15.2.1.2	D_3	PASS, NO_OPTION
2002	15.2.1.2	R_1	Reference Assertion, no test results
2003	15.2.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
2004	15.2.1.2	11	PASS
2005	15.2.1.2	12	PASS
2006	15.2.1.2	13	PASS, NO_OPTION
2007	15.2.1.2	14	PASS, NO_OPTION, NO_TEST_SUPPORT
2008	15.2.1.2	15	PASS, NO_OPTION, NO_TEST_SUPPORT
2009	15.2.1.2	16	PASS, NO_OPTION, NO_TEST_SUPPORT
2010	15.2.1.2	17	PASS, NO_OPTION, NO_TEST_SUPPORT
2011	15.2.1.2	18	PASS, NO_OPTION, NO_TEST_SUPPORT
2012	15.2.1.2	19	PASS, NO_OPTION, NO_TEST_SUPPORT
2013	15.2.1.2	20	PASS, NO_OPTION, NO_TEST_SUPPORT
2014	15.2.1.2	21	PASS, NO_OPTION, NO_TEST_SUPPORT
2015	15.2.1.2	22	PASS, NO_OPTION, NO_TEST_SUPPORT
2016	15.2.1.2	23	PASS, NO_OPTION, NO_TEST_SUPPORT
2017	15.2.1.2	24	PASS, NO_OPTION, NO_TEST_SUPPORT
2018	15.2.1.2	D_4	PASS, NO_OPTION
2019	15.2.1.2	D_5	PASS, NO_OPTION
2020	15.2.1.2	25	PASS, NO_OPTION, NO_TEST_SUPPORT
2021	15.2.1.2	R_2	Reference Assertion, no test results
2022	15.2.1.2	R_3	Reference Assertion, no test results
2023	15.2.1.2	26	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
2024	15.2.1.2	27	PASS, NO_OPTION, NO_TEST_SUPPORT
2025	15.2.1.2	28	PASS, NO_OPTION, NO_TEST_SUPPORT
2026	15.2.1.2	D_7	PASS, NO_OPTION
2027	15.2.1.3	R_4	Reference Assertion, no test results
2028	15.2.1.3	R_5	Reference Assertion, no test results
2029	15.2.1.4	mq_open_EACCESS1	PASS, NO_OPTION, NO_TEST_SUPPORT
2030	15.2.1.4	mq_open_EACCESS2	PASS, NO_OPTION, NO_TEST_SUPPORT
2031	15.2.1.4	mq_open_EEXIST	PASS, NO_OPTION, NO_TEST_SUPPORT



2032	15.2.1.4	29	PASS, NO_OPTION, NO_TEST_SUPPORT
2033	15.2.1.4	30	PASS, NO_OPTION, NO_TEST_SUPPORT
2034	15.2.1.4	D_8	PASS, NO_OPTION
2035	15.2.1.4	31	PASS, NO_OPTION, NO_TEST_SUPPORT
2036	15.2.1.4	32	PASS, NO_OPTION, NO_TEST_SUPPORT
2037	15.2.1.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT
2038	15.2.1.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
2039	15.2.1.4	35	PASS, NO_OPTION, NO_TEST_SUPPORT
2040	15.2.1.4	36	PASS, NO_OPTION, NO_TEST_SUPPORT
2041	15.2.1.4	37	PASS, NO_OPTION, NO_TEST_SUPPORT
2042	15.2.1.4	mq_open_ENOENT	PASS, NO_OPTION, NO_TEST_SUPPORT
2043	15.2.1.4	38	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
2044	15.2.1.4	39	PASS, NO_OPTION

---

2045 **Assertions for mq\_close**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
2047	15.2.2.1	1	PASS[1, 2], NO_OPTION
2048	15.2.2.1	2	PASS[1, 2], NO_OPTION
2049	15.2.2.1	3	PASS, NO_OPTION
2050	15.2.2.1	4	PASS, NO_OPTION
2051	15.2.2.2	mq_close	PASS, NO_OPTION
2052	15.2.2.2	D_1	PASS, NO_OPTION
2053	15.2.2.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
2054	15.2.2.2	D_2	PASS, NO_OPTION
2055	15.2.2.3	R_1	Reference Assertion, no test results
2056	15.2.2.3	R_2	Reference Assertion, no test results
2057	15.2.2.4	6	PASS, NO_OPTION
2058	15.2.2.4	7	PASS, NO_OPTION

---

2059 **Assertions for mq\_unlink**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
2061	15.2.3.1	1	PASS[1, 2], NO_OPTION
2062	15.2.3.1	2	PASS[1, 2], NO_OPTION
2063	15.2.3.1	3	PASS, NO_OPTION
2064	15.2.3.1	4	PASS, NO_OPTION
2065	15.2.3.2	mq_unlink	PASS, NO_OPTION
2066	15.2.3.2	R_1	Reference Assertion, no test results
2067	15.2.3.2	5	PASS, NO_OPTION
2068	15.2.3.2	D_1	PASS, NO_OPTION
2069	15.2.3.3	R_2	Reference Assertion, no test results
2070	15.2.3.3	R_3	Reference Assertion, no test results
2071	15.2.3.4	6	PASS, NO_OPTION
2072	15.2.3.4	7	PASS, NO_OPTION, NO_TEST_SUPPORT
2073	15.2.3.4	mq_unlink_ENOENT	PASS, NO_OPTION
2074	15.2.3.4	8	PASS, NO_OPTION

---

2075 **Assertions for mq\_send**

2076	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
2077	15.2.4.1	1	PASS[1, 2], NO_OPTION
2078	15.2.4.1	2	PASS[1, 2], NO_OPTION
2079	15.2.4.1	3	PASS, NO_OPTION
2080	15.2.4.1	4	PASS, NO_OPTION
2081	15.2.4.2	mq_send	PASS, NO_OPTION
2082	15.2.4.2	R_1	Reference Assertion, no test results
2083	15.2.4.2	5	PASS, NO_OPTION
2084	15.2.4.2	6	PASS, NO_OPTION
2085	15.2.4.2	7	PASS, NO_OPTION
2086	15.2.4.2	R_2	Reference Assertion, no test results
2087	15.2.4.2	8	PASS, NO_OPTION
2088	15.2.4.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
2089	15.2.4.2	D_1	PASS, NO_OPTION
2090	15.2.4.3	R_4	Reference Assertion, no test results
2091	15.2.4.3	R_5	Reference Assertion, no test results
2092	15.2.4.4	mq_send_EAGAIN	PASS, NO_OPTION
2093	15.2.4.4	10	PASS, NO_OPTION
2094	15.2.4.4	11	PASS, NO_OPTION
2095	15.2.4.4	mq_send_EINVAL	PASS, NO_OPTION, NO_TEST_SUPPORT
2096	15.2.4.4	mq_send_MSGSIZE	PASS, NO_OPTION
2097	15.2.4.4	12	PASS, NO_OPTION

2098 **Assertions for mq\_receive**

2099	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
2100	15.2.5.1	1	PASS[1, 2], NO_OPTION
2101	15.2.5.1	2	PASS[1, 2], NO_OPTION
2102	12.2.5.1	3	PASS, NO_OPTION
2103	12.2.5.1	4	PASS, NO_OPTION
2104	12.2.5.2	mq_receive	PASS, NO_OPTION
2105	12.2.5.2	5	PASS, NO_OPTION
2106	12.2.5.2	6	PASS, NO_OPTION
2107	12.2.5.2	7	PASS, NO_OPTION
2108	12.2.5.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
2109	12.2.5.2	D_1	PASS, NO_OPTION
2110	12.2.5.2	R_1	Reference Assertion, no test results
2111	12.2.5.2	D_2	PASS, NO_OPTION
2112	12.2.5.3	R_2	Reference Assertion, no test results
2113	12.2.5.3	R_3	Reference Assertion, no test results
2114	12.2.5.4	mq_receive_EAGAIN	PASS, NO_OPTION
2115	12.2.5.4	9	PASS, NO_OPTION
2116	12.2.5.4	10	PASS, NO_OPTION
2117	12.2.5.4	11	PASS, NO_OPTION
2118	12.2.5.4	12	PASS, NO_OPTION
2119	12.2.5.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT

2120 **Assertions for mq\_notify**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
2121			
2122	15.2.6.1	1	PASS[1, 2], NO_OPTION
2123	15.2.6.1	2	PASS[1, 2], NO_OPTION
2124	15.2.6.1	3	PASS, NO_OPTION
2125	15.2.6.1	4	PASS, NO_OPTION
2126	15.2.6.2	mq_notify	PASS, NO_OPTION
2127	15.2.6.2	R_1	Reference Assertion, no test results
2128	15.2.6.2	5	PASS, NO_OPTION
2129	15.2.6.2	6	PASS, NO_OPTION
2130	15.2.6.2	7	PASS, NO_OPTION
2131	15.2.6.2	D_1	PASS, NO_OPTION
2132	15.2.6.3	R_2	Reference Assertion, no test results
2133	15.2.6.3	R_3	Reference Assertion, no test results
2134	15.2.6.4	8	PASS, NO_OPTION
2135	15.2.6.4	mq_notify_EBUSY	PASS, NO_OPTION
2136	15.2.6.4	9	PASS, NO_OPTION

2137 **Assertions for mq\_setattr**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
2138			
2139	15.2.7.1	1	PASS[1, 2], NO_OPTION
2140	15.2.7.1	2	PASS[1, 2], NO_OPTION
2141	15.2.7.1	3	PASS, NO_OPTION
2142	15.2.7.1	4	PASS, NO_OPTION
2143	15.2.7.2	mq_setattr	PASS, NO_OPTION
2144	15.2.7.2	5	PASS, NO_OPTION
2145	15.2.7.2	D_1	PASS, NO_OPTION
2146	15.2.7.2	6	PASS, NO_OPTION
2147	15.2.7.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
2148	15.2.7.2	D_2	PASS, NO_OPTION
2149	15.2.7.3	R_1	Reference Assertion, no test results
2150	15.2.7.3	R_2	Reference Assertion, no test results
2151	15.2.7.4	8	PASS, NO_OPTION
2152	15.2.7.4	9	PASS, NO_OPTION

2153 **Assertions for mq\_getattr**

	<b>Subclause</b>	<b>Assertion ID</b>	<b>Conforming Results</b>
2154			
2155	15.2.8.1	1	PASS[1, 2], NO_OPTION
2156	15.2.8.1	2	PASS[1, 2], NO_OPTION
2157	15.2.8.1	3	PASS, NO_OPTION
2158	15.2.8.1	4	PASS, NO_OPTION
2159	15.2.8.2	mq_getattr	PASS, NO_OPTION
2160	15.2.8.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
2161	15.2.8.2	6	PASS, NO_OPTION
2162	15.2.8.2	7	PASS, NO_OPTION

2163	15.2.8.2	D_1	PASS, NO_OPTION
2164	15.2.8.3	R_1	Reference Assertion, no test results
2165	15.2.8.3	R_2	Reference Assertion, no test results
2166	15.2.8.3	8	PASS, NO_OPTION
2167	15.2.8.4	9	PASS, NO_OPTION
2168	15.2.8.4	10	PASS, NO_OPTION

---

## Annex B (informative) Bibliography

180 This Annex contains lists of related open systems standards and suggested reading on historical implementations  
181 and application programming.

### 182 **B.1 Related Open Systems Standards**

#### 183 **B.1.1. Networking Standards**

184 {B1} ISO 7498: 1984, *Information processing systems – Open Systems Interconnection – Basic Reference Model*.  
185

186 {B2} ISO 8072: 1986, *Information processing systems – Open Systems Interconnection – Transport service*  
187 *definition*.

188 {B3} ISO/IEC 8073: 1988, *Information processing systems – Open Systems Interconnection – Connection oriented*  
189 *transport protocol specification*.<sup>5)</sup>

190 {B4} ISO 8326: 1987, *Information processing systems – Open Systems Interconnection – Basic connection oriented*  
191 *session service definition*.

192 {B5} ISO 8327: 1987, *Information processing systems – Open Systems Interconnection – Basic connection oriented*  
193 *session protocol definition*.

194 {B6} ISO 8348: 1987, *Information processing systems – Data communications – Network service definition*.

21 {B7} ISO 8473: 1988, *Information processing systems – Data communications – Protocol for providing the*  
22 *connectionless-mode network service*.

23 {B8} ISO 8571: 1988, *Information processing systems – Open Systems Interconnection – File Transfer, Access and*  
24 *Management*.

25 {B9} ISO 8649: 1988, *Information processing systems – Open Systems Interconnection – Service definition for the*  
26 *Association Control Service Element*.

27 {B10} ISO 8650: 1988, *Information processing systems – Open Systems Interconnection – Protocol*  
28 *specification for the Association Control Service Element*.

---

4)

17 ISO documents can be obtained from the ISO office, 1, rue de Varembe, Case Postale 56, CH-1211, Genève 20,  
18 Switzerland/Suisse.

5)

19 IEC documents can be obtained from the IEC office, 3, rue de Varembe, Case Postale 131, CH-1211, Genève 20,  
20 Switzerland/Suisse.

- 29 {B11} ISO 8802-2: 1989 [IEEE Std 802.2-1989 (ANSI)], *Information processing systems – Local area networks*  
30 *– Part 2: Logical link control.*
- 31 {B12} ISO 8802-3: 1989 [IEEE 802.3-1988 (ANSI)], *Information processing systems – Local area networks –*  
32 *Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical*  
33 *layer specifications.*
- 34 {B13} ISO/IEC 8802-4: 1990 [IEEE Std 802.4-1990 (ANSI)], *Information technology – Local area networks – Part*  
35 *4: Token-passing bus access method and physical layer specifications.*
- 36 {B14} ISO 8802\_5: ... (IEEE 802.5-1989), *Information technology – Local area networks – Part 5: Token ring*  
37 *access method and physical layer specifications.*
- 38 {B15} ISO 8822: 1988, *Information processing systems – Open Systems Interconnection – Connection oriented*  
39 *presentation service definition.*
- 40 {B16} ISO 8823: 1988, *Information processing systems – Open Systems Interconnection – Connection oriented*  
41 *presentation protocol specification.*
- 42 {B17} ISO 8831: 1989, *Information processing systems – Open Systems Interconnection – Job transfer and*  
43 *manipulation concepts and services.*
- 44 {B18} ISO 8832: 1989, *Information processing systems – Open Systems Interconnection – Specification of the*  
45 *basic class protocol for job transfer and manipulation.*
- 46 {B19} CCITT Recommendation X.25, *Interface between data terminal equipment (DTE) and data circuit-*  
47 *terminating equipment (DCE) for terminals operating in the packet mode and connected to public data*  
48 *networks by dedicated circuit.<sup>6)</sup>*
- 49 {B20} CCITT Recommendation X.212, *Information processing systems – Data communication – Data link*  
50 *service definition for Open Systems Interconnection.*
- 54 **B.1.2 Language Standards**
- 55 {B21} ISO 1539: 1980, *Programming languages – FORTRAN.*
- 56 {B22} ISO 1989: 1985, *Programming Languages – COBOL.*
- 57
- 58 {B23} ISO 8652: 1987, *Programming Languages – Ada.*
- 59 {B24} ANSI X3.113\_1987<sup>7)</sup>, *Information systems – Programming language – FULL BASIC.*
- 60 {B25} ANSI/IEEE 770X3.97-1983, *Standard Pascal Computer Programming Language.*
- 61 {B26} ANSI/MDC X11.1-1984, *Programming Language MUMPS.*
- 62 **B.1.3 Graphics Standards**
- 63 {B27} ISO 7942: 1985, *Information processing systems – Computer graphics – Graphical Kernel System (GKS)*  
64 *functional description.*

---

6)

52 CCITT documents can be obtained from the CCITT General Secretariat, International Telecommunications Union, Sales  
53 Section, Place des Nations, CH-1211, Genève 20, Switzerland/Suisse.

7)

78 ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway,  
79 New York, NY 10018.

65 {B28} ISO 8632: 1987, *Information processing systems – Computer graphics – Metafile for the storage and*  
66 *transfer of picture description information.*

67 {B29} ISO/IEC 9592: 1989 (ANSI X3.144-1988), *Information processing systems – Computer graphics –*  
68 *Programmer’s hierarchical interactive graphics system (PHIGS).*

#### 69 **B.1.4 Database Standards**

70 {B30} ISO 8907: 1987, *Database Language – NDL.*

71 {B31} ISO 9075: 1987, *Database Language – SQL.*

## 72 **B.2 Other Standards**

73 {B32} ISO 639: 1988, *Code for the representation of names of languages.*

74 {B33} ISO 3166: 1988, *Code for the representation of names of countries.*

75 {B34} ISO 8859-1: 1987, *Information Processing – 8-bit single-byte coded graphic character sets – Part 1:*  
76 *Latin alphabet No. 1.*

77 {B35} ISO 9127: 1988, *Information processing systems – User documentation and cover information for*  
78 *consumer software packages.*

80 {B36} ISO/IEC 9945-2: ...,<sup>8)</sup> *Information technology – Portable operating system interface (POSIX) – Part 2:*  
81 *Shell and utilities.*

82 {B37} ISO/IEC 10646:...,<sup>9)</sup> *Information processing – Multiple octet coded character set.*

83 {B38} IEEE Std 100-1988, *IEEE Standard Dictionary of Electrical and Electronics Terms.*

84 {B39} P1003.0/D16,<sup>10)</sup> *Draft Guide to the POSIX Open Systems Environment.*

85 {B40} ISO/IEC TR 10000-1: 1990, *Information technology – Framework and taxonomy of International*  
86 *Standardized Profiles – Part 1: Framework.*

## 87 **B.3 Historical Documentation and Introductory Texts**

88 {B41} American Telephone and Telegraph Company, *System V Interface Definition (SVID), Issues 2 and 3.*  
89 *Morristown, NJ: UNIX Press, 1986, 1989.<sup>11)</sup>*

90 {B42} American Telephone and Telegraph Company. *UNIX System III Programmer’s Manual.* Greensboro,  
91 *NC: Western Electric Company, October 1981.*

92 {B43} American Telephone and Telegraph Company. *UNIX Time Sharing System: UNIX Programmer’s*  
93 *Manual.* 7<sup>th</sup> ed. Murray Hill, NJ: Bell Laboratories, January 1979.

---

102 <sup>8)</sup> To be approved and published.

103 <sup>9)</sup> To be approved and published.

104 <sup>10)</sup> This unapproved draft document is available from IEEE Publications, 445 Hoes Lane. P.O. Box 1331, Piscataway, NJ 0055-  
105 1331. Telephone: 1(800) 678 -IEEE or +1 (908) 981-1393 (outside US).

106 <sup>11)</sup> This is one of several documents that represent an industry specification in an area related to POSIX.1. The creators of such  
107 documents may be able to identify newer versions that may be interesting.

- 94 {B44} “The UNIX System.”<sup>12)</sup> *AT&T Bell Laboratories Technical Journal*. vol. 63 (8 Part 2), October 1984.
- 95 {B45} “UNIX Time-Sharing System.”<sup>13)</sup> *Bell System Technical Journal*. vol. 57 (6 Part 2), July-August 1978.
- 96 {B46} Bach, Maurice J. *The Design of the UNIX Operating System*. Englewood Cliffs, NJ: Prentice-Hall, 1987.
- 97 {B47} Dijkstra, E.W. “Solution of a Problem in Concurrent Programming Control,” *Communications of the ACM*, vol. 8(9), September 1965, pp. 569-570.
- 110 {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.
- 111
- 112 {B49} Harbison, Samuel P. and Steele, Guy L. *C: A Reference Manual*. Englewood Cliffs, NJ: Prentice\_Hall, 1987.
- 113
- 114 {B50} Kernighan, Brian W. and Ritchie, Dennis M. *The C Programming Language*. Englewood Cliffs, NJ: Prentice-Hall, 1978.
- 115
- 116 {B51} Kernighan, Brian W. and Pike, Rob. *The UNIX Programming Environment*. Englewood Cliffs, NJ: Prentice-Hall, 1984.
- 117
- 118 {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.
- 119
- 120
- 121 {B53} McGilton, Henry and Morgan, Rachel. *Introducing the UNIX System*. New York: McGraw-Hill (BYTE BOOKS), 1983.
- 122
- 123 {B54} Organick, Elliot I. *The Multics System: An Examination of Its Structure*. Cambridge, MA: the MIT Press, 1972.
- 124
- 125 {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.
- 126
- 127 {B56} Ritchie, Dennis M. “Reflections on Software Research.” *Communications of the ACM*, vol. 27(8), August 1984, pp. 758-760. ACM Turing Award Lecture.
- 128
- 129 {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.
- 130
- 131 {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.
- 132
- 133 {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.
- 134
- 135 {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.
- 136
- 137 {B61} Rochkind, Marc J. *Advanced UNIX Programming*. Englewood Cliffs, NJ: Prentice-Hall, 1985.
- 138 {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.
- 139
- 140

---

108 <sup>12)</sup> This entire edition is devoted to the UNIX system.

109 <sup>13)</sup> This entire edition is devoted to the UNIX time-sharing system.



- 141 {B63} /usr/group Standards Committee. *1984 /usr/group Standard*. Santa Clara, CA: Uni\_Forum, 1984.
- 142 {B64} X/Open Company, Ltd. *X/Open Portability Guide, Issue 2*. Amsterdam: Elsevier Science Publishers,  
143 1987.
- 144 {B65} X/Open Company, Ltd. *X/Open Portability Guide, Issue 3*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

145 **B.4 Other Sources of Information**

- 146 {B66} ISO/IEC JTC 1 N1335, *Final Report of ISO/IEC JTC 1 TSG-1 on Standards Necessary to Define Interfaces*  
147 *for Application Portability (IAP)*.

This page is intentionally blank.

## Identifier Index

<i>access()</i>	Check File Accessibility {5.6.3} . . . . .	92
<i>aio_cancel()</i>	Cancel Asynchronous I/O Request {6.7.7} . . . . .	144
<i>aio_cb()</i>	Asynchronous I/O Control Block {6.7.1.1} . . . . .	117
<i>aio_error()</i>	Retrieve Error Status of Asynchronous I/O Operation {6.7.5} . . . . .	141
<i>aio_fsync()</i>	Asynchronous File Synchronization {6.7.9} . . . . .	149
<i>&lt;aio.h&gt;</i>	Asynchronous I/O Control Block {6.7.1.1} . . . . .	114
<i>aio_read()</i>	Asynchronous Read {6.7.2} . . . . .	117
<i>aio_return()</i>	Retrieve Return Status of Asynchronous I/O Operation {6.7.6} . . . . .	142
<i>aio_suspend()</i>	Wait for Asynchronous I/O Request {6.7.8} . . . . .	146
<i>aio_write()</i>	Asynchronous Write {6.7.3} . . . . .	123
<i>alarm()</i>	Schedule Alarm {3.4.1} . . . . .	79
<i>chdir()</i>	Change Current Working Directory {5.2.1} . . . . .	85
<i>chmod()</i>	Change File Modes {5.6.4} . . . . .	92
<i>chown()</i>	Change Owner and Group of a File {5.6.5} . . . . .	95
<i>clock_getres()</i>	Clocks {14.2.1} . . . . .	257
<i>clock_gettime()</i>	Clocks {14.2.1} . . . . .	257
<i>clockid_t</i>	Type Definitions {14.1.3} . . . . .	256
<i>clock_settime()</i>	Clocks {14.2.1} . . . . .	257
<i>close()</i>	Close a File {6.3.1} . . . . .	103
<i>closedir()</i>	Directory Operations {5.1.2} . . . . .	85
<i>creat()</i>	Create a New File or Rewrite an Existing One {5.3.2} . . . . .	87
<i>dev_t</i>	Primitive System Data Types {2.5} . . . . .	39
<i>directory</i>	Directory Operations {5.1.2} . . . . .	85
<i>dup()</i>	Duplicate an Open File Descriptor {6.2.1} . . . . .	102
<i>dup2()</i>	Duplicate an Open File Descriptor {6.2.1} . . . . .	102
<i>&lt;errno.h&gt;</i>	Error Numbers {2.4} . . . . .	38
<i>execl()</i>	Execute a File {3.1.2} . . . . .	54
<i>execle()</i>	Execute a File {3.1.2} . . . . .	54
<i>execlp()</i>	Execute a File {3.1.2} . . . . .	54
<i>execv()</i>	Execute a File {3.1.2} . . . . .	54
<i>execve()</i>	Execute a File {3.1.2} . . . . .	54
<i>execvp()</i>	Execute a File {3.1.2} . . . . .	54
<i>_exit()</i>	Terminate a Process {3.2.2} . . . . .	57
<i>fchmod()</i>	Change File Modes {5.6.4} . . . . .	92
<i>fcntl()</i>	File Control {6.5.2} . . . . .	106
<i>&lt;fcntl.h&gt;</i>	Data Definitions for File Control Operations {6.5.1} . . . . .	106
<i>fdatasync()</i>	Synchronize the Data of a File {6.6.2} . . . . .	112
<i>fdopen()</i>	Open a Stream on a File Descriptor {8.2.2} . . . . .	157
<i>fork()</i>	Process Creation {3.1.1} . . . . .	52
<i>fpathconf()</i>	Get Configurable Pathname Variables {5.7.1} . . . . .	98
<i>fstat()</i>	Get File Status {5.6.2} . . . . .	91
<i>fsync()</i>	Synchronize the State of a File {6.6.1} . . . . .	110
<i>ftruncate()</i>	Truncate a File to a Specified Length {5.6.7} . . . . .	96
<i>getcwd()</i>	Get Working Directory Pathname {5.2.2} . . . . .	86
<i>gid_t</i>	Primitive System Data Types {2.5} . . . . .	39
<i>ino_t</i>	Primitive System Data Types {2.5} . . . . .	39
<i>kill()</i>	Send a Signal to a Process {3.3.2} . . . . .	68
<i>&lt;limits.h&gt;</i>	Minimum Values {2.8.2} . . . . .	44
<i>link()</i>	Link to a File {5.3.4} . . . . .	88
<i>lio_listio()</i>	List Directed I/O {6.7.4} . . . . .	130
<i>lseek()</i>	Reposition Read/Write File Offset {6.5.3} . . . . .	109
<i>mkdir()</i>	Make a Directory {5.4.1} . . . . .	88
<i>mkfifo()</i>	Make a FIFO Special File {5.4.2} . . . . .	88
<i>mlock()</i>	Lock/Unlock a Range of Process Address Space {12.1.2} . . . . .	198
<i>mlockall()</i>	Lock/Unlock the Address Space of a Process {12.1.1} . . . . .	194
<i>mmap()</i>	Map Process Addresses to a Memory Object {12.2.1} . . . . .	203
<i>mode_t</i>	Primitive System Data Types {2.5} . . . . .	39

<i>mprotect()</i>	Change Memory Protection {12.2.3} . . . . .	215
<i>mq_attr()</i>	Data Structures {15.1.1} . . . . .	280
<i>mq_close()</i>	Close a Message Queue {15.2.2} . . . . .	289
<i>mqd_t()</i>	Data Structures {15.1.1} . . . . .	280
<i>mq_getattr()</i>	Get Message Queue Attributes {15.2.8} . . . . .	302
<i>mq_notify()</i>	Notify Process that a Message is Available on a Queue {15.2.6} . . . . .	298
<i>mq_open()</i>	Open a Message Queue {15.2.1} . . . . .	281
<i>mq_receive()</i>	Receive a Message from a Message Queue {15.2.5} . . . . .	295
<i>mq_send()</i>	Send a Message to a Message Queue {15.2.4} . . . . .	293
<i>mq_setattr()</i>	Set Message Queue Attributes {15.2.7} . . . . .	300
<mqqueue.h>	Data Structures {15.1.1} . . . . .	280
<i>mq_unlink()</i>	Remove a Message Queue {15.2.3} . . . . .	291
<i>msync()</i>	Memory Object Synchronization {12.2.4} . . . . .	220
<i>munlock()</i>	Lock/Unlock a Range of Process Address Space {12.1.2} . . . . .	198
<i>munlockall()</i>	Lock/Unlock the Address Space of a Process {12.1.1} . . . . .	194
<i>munmap()</i>	Unmap Previously Mapped Addresses {12.2.2} . . . . .	213
<i>nanosleep()</i>	High Resolution Sleep {14.2.5} . . . . .	275
<i>nlink_t()</i>	Primitive System Data Types {2.5} . . . . .	39
<i>off_t()</i>	Primitive System Data Types {2.5} . . . . .	39
<i>open()</i>	Open a File {5.3.1} . . . . .	86
<i>opendir()</i>	Directory Operations {5.1.2} . . . . .	85
<i>pathconf()</i>	Get Configurable Pathname Variables {5.7.1} . . . . .	98
<i>pause()</i>	Suspend Process Execution {3.4.2} . . . . .	79
<i>pid_t()</i>	Primitive System Data Types {2.5} . . . . .	39
<i>pipe()</i>	Create an Inter-Process Channel {6.1.1} . . . . .	102
<i>read()</i>	Read from a File {6.4.1} . . . . .	104
<i>readdir()</i>	Directory Operations {5.1.2} . . . . .	85
<i>rename()</i>	Rename a File {5.5.3} . . . . .	89
<i>rewinddir()</i>	Directory Operations {5.1.2} . . . . .	85
<i>rmdir()</i>	Remove a Directory {5.5.2} . . . . .	89
<i>sched_getparam()</i>	Get Scheduling Parameters {13.3.2} . . . . .	242
<i>sched_get_priority_max()</i>	Get Scheduling Parameter Limits {13.3.6} . . . . .	250
<i>sched_get_priority_min()</i>	Get Scheduling Parameter Limits {13.3.6} . . . . .	250
<i>sched_getscheduler()</i>	Get Scheduling Policy {13.3.4} . . . . .	248
<sched.h>	Scheduling Parameters {13.1} . . . . .	235
<i>sched_param</i>	Scheduling Parameters {13.1} . . . . .	235

<i>sched_rr_get_interval()</i>	Get Scheduling Parameter Limits {13.3.6} .....	251
<i>sched_setparam()</i>	Set Scheduling Parameters {13.3.1} .....	239
<i>sched_setscheduler()</i>	Set Scheduling Policy and Scheduling Parameters {13.3.3} .....	244
<i>sched_yield()</i>	Yield Processor {13.3.5} .....	249
<semaphore.h>	Semaphore Characteristics {11.1} .....	160
<i>sem_close()</i>	Close a Named Semaphore {11.2.4} .....	171
<i>sem_destroy()</i>	Destroy an Unnamed Semaphore {11.2.2} .....	164
<i>sem_getvalue()</i>	Get the Value of a Semaphore {11.2.8} .....	187
<i>sem_init()</i>	Initialize an Unnamed Semaphore {11.2.1} .....	161
<i>sem_open()</i>	Initialize/Open a Named Semaphore {11.2.3} .....	166
<i>sem_post()</i>	Unlock a Semaphore {11.2.7} .....	183
<i>sem_t</i>	Semaphore Characteristics {11.1} .....	160
<i>sem_trywait()</i>	Lock a Semaphore {11.2.6} .....	175
<i>sem_unlink()</i>	Remove a Named Semaphore {11.2.5} .....	173
<i>sem_wait()</i>	Lock a Semaphore {11.2.6} .....	175
<i>shm_open()</i>	Open a Shared Memory Object {12.3.2} .....	224
<i>shm_unlink()</i>	Remove a Shared Memory Object {12.3.2} .....	231
<i>sigaction()</i>	Examine and Change Signal Action {3.3.4} .....	69
<i>sigaddset()</i>	Manipulate Signal Sets {3.3.3} .....	69
<i>sigdelset()</i>	Manipulate Signal Sets {3.3.3} .....	69
<i>sigemptyset()</i>	Manipulate Signal Sets {3.3.3} .....	69
<i>sigevent()</i>	Signal Generation and Delivery {3.3.1.2} .....	60, 61
<i>sigfillset()</i>	Manipulate Signal Sets {3.3.3} .....	69
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	64
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	64
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	64
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	65
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	65
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	65
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	65
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	65
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	65
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	65
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	66
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	70
<i>siginfo_t</i>	Signal Actions {3.3.1.3} .....	66
<i>sigismember()</i>	Manipulate Signal Sets {3.3.3} .....	69
<i>sigpending()</i>	Examine Pending Signals {3.3.6} .....	71
<i>sigprocmask()</i>	Examine and Change Blocked Signals {3.3.5} .....	70
<i>sigqueue()</i>	Queue a Signal to a Process {3.3.9} .....	75
<i>sigsetops()</i>	Manipulate Signal Sets {3.3.3} .....	69
<i>sigsuspend()</i>	Wait for a Signal {3.3.7} .....	71
<i>sigtimedwait()</i>	Synchronously Accept a Signal {3.3.8} .....	72
<i>sigwaitinfo()</i>	Synchronously Accept a Signal {3.3.8} .....	72
<i>size_t</i>	Primitive System Data Types {2.5} .....	39
<i>sleep()</i>	Delay Process Execution {3.4.3} .....	80
<i>ssize_t</i>	Primitive System Data Types {2.5} .....	39
<i>stat()</i>	Get File Status {5.6.2} .....	91
<i>sysconf()</i>	Get Configurable System Variables {4.8.1} .....	81
<sys/stat.h>	File Characteristics: Header and Data Structure {5.6.1} .....	90
<sys/types.h>	Primitive System Data Types {2.5} .....	39
<sys/wait.h>	Wait for Process Termination {3.2.1} .....	57
<time.h>	Time Value Specification Structures {14.1.1} .....	255
<time.h>	Time Value Specification Structures {14.1.1} .....	255
<i>timer_create()</i>	Create a Per-Process Timer {14.2.2} .....	263
<i>timer_delete()</i>	Delete a Per-Process Timer {14.2.3} .....	267
<i>timer_getoverrun()</i>	Per-Process Timers {14.2.4} .....	269

<i>timer_gettime()</i>	Per-Process Timers {14.2.4} . . . . .	269
<i>timer_settime()</i>	Per-Process Timers {14.2.4} . . . . .	269
<i>timer_t</i>	Type Definitions {14.1.3} . . . . .	256
<i>time_t</i>	Primitive System Data Types {2.5} . . . . .	39
<i>uid_t</i>	Primitive System Data Types {2.5} . . . . .	39
<i>umask()</i>	Set File Creation Mask {5.3.3} . . . . .	87
<i>unlink()</i>	Remove Directory Entries {5.5.1} . . . . .	89
<i>utime()</i>	Set File Creation Mask {5.5.3} . . . . .	96
<i>wait</i>	Wait for Process Termination {3.2.1} . . . . .	57
<i>waitpid()</i>	Wait for Process Termination {3.2.1} . . . . .	57
<i>write()</i>	Write to a File {6.4.2} . . . . .	105

## Alphabetic Topical Index

mq_close()	
function definition . . . . .	289
pathconf()	
function definition . . . . .	98
rename()	
function definition . . . . .	89
sched_getparam()	
function definition . . . . .	242
(time) resolution	
definition of . . . . .	28
< aio.h >	
header definition . . . . .	114
<fcntl.h>	
header definition . . . . .	106
<mqqueue.h>	
header definition . . . . .	280
< sched.h >	
header definition . . . . .	235
< semaphore.h >	
header definition . . . . .	160
< sys/stat.h >	
File Modes . . . . .	91
File Types . . . . .	90
header definition . . . . .	90
Time Entries . . . . .	91
< time.h >	
header definition . . . . .	255
_exit	
function definition . . . . .	57
_GAP_ . . . . .	10
_RAP_ . . . . .	10
_sc_aio_listio_max	
limit definition . . . . .	83
_sc_aio_max	
limit definition . . . . .	83
_sc_aio_prio_delta_max	
limit definition . . . . .	83
_sc_asynchronous_io	
limit definition . . . . .	83
_sc_delaytimer_max	
limit definition . . . . .	83
_sc_fsync	
limit definition . . . . .	83
_sc_mapped_files	
limit definition . . . . .	83
_sc_memlock	
limit definition . . . . .	83
_sc_memlock_range	
limit definition . . . . .	83
_sc_memory_protection	
limit definition . . . . .	83
_sc_message_passing	
limit definition . . . . .	83
_sc_mq_open_max	
limit definition . . . . .	83
_sc_mq_prio_max	
limit definition . . . . .	83
_sc_pagesize	
limit definition . . . . .	83
_sc_prioritized_io	
limit definition . . . . .	83
_sc_priority_scheduling	
limit definition . . . . .	83
_sc_realtime_signals	
limit definition . . . . .	83
_sc_sem_nsems_max	
limit definition . . . . .	83
_sc_sem_value_max	
limit definition . . . . .	83
_sc_semaphores	
limit definition . . . . .	83
_sc_shared_memory_objects	
limit definition . . . . .	83
_sc_sigqueue_max	
limit definition . . . . .	83
_sc_synchronized_io	
limit definition . . . . .	83
_sc_timer_max	
limit definition . . . . .	83
_sc_timers	
limit definition . . . . .	83
Abbreviations . . . . .	9
absolute pathname	
definition of . . . . .	24
access mode	
definition of . . . . .	24
access()	
function definition . . . . .	92
ACM . . . . .	356
address space	
definition of . . . . .	24
aio_alldone	
definition of . . . . .	117
aio_cancel()	
function definition . . . . .	144
aio_canceled . . . . .	117
aio_error()	
function definition . . . . .	141
aio_fsync()	
function definition . . . . .	149

aio_listio_max	81, 83
aio_max	81, 83, 121, 127, 134, 135, 153
aio_notcanceled	
definition of	117
aio_prio_delta_max	81, 83
aio_read()	
function definition	117
aio_return()	
function definition	142
aio_suspend()	
function definition	146
aio_write()	
function definition	123
aiocb	
C-language type definition	114
alarm	
schedule	79
alarm()	
function definition	79
ANSI	354, 355
ANSI/IEEE	354
ANSI/MDC	354
appropriate privileges	11, 14, 24, 32, 51, 94, 95, 164, 203, 246
arm (a timer)	
definition of	24
Asynchronous File Synchronization	149
asynchronous I/O completion	
definition of	24
Asynchronous I/O Control Block	114
asynchronous I/O operation	
definition of	24
Asynchronous Input and Output	114
Asynchronous Read	117
Asynchronous Write	123
AT&T	356
background	24, 60, 129, 136, 139
background process	
definition of	24
background process group	
definition of	24
BASIC	353, 354
Bibliography	1, 353
blocked process	
definition of	24
BSD	
4.2	356
4.3	356
BYTE	355, 356
C language	
Language -Dependent Services	7
C Standard	
Abbreviation	9
definition of	9
C Standard Language-Dependent System Support	7
C-language type definitions	
aiocb	114
clockid_t	256
mq_attr	280
mqd_t	280
sched_param	235
sem_t	160
sigevent	60
siginfo_t	61, 64-66, 70, 72
timer_t	256
Cancel Asynchronous i/o Request	144
CCITT	354
Change Current Working Directory	85
Change File Modes	92
Change Memory Protection	215
Change Owner and Group of a File	95
character	
definition of	24
character special file	
definition of	24
chdir()	
function definition	85
Check File Accessibility	92
child process	
definition of	24
chmod()	
function definition	92
chown()	
function definition	95
CH-1211	6, 353, 354
clk_tck	84
definition of	84
clock	
definition of	24
clock tick	83
definition of	24
clock_getres	
Reference Assertion definition	259
clock_getres()	
function definition	257
clock_gettime	
Reference Assertion definition	259
clock_gettime()	
function definition	257
clock_realtime	276
definition of	256
clock_settime	
Reference Assertion definition	259
clock_settime()	
function definition	257
clockid_t	
C-language type definition	256
Clocks	257



Clocks and Timer Functions . . . . .	257
Clocks and Timers . . . . .	255, 342
definition of . . . . .	255
Close a File . . . . .	103
Close a Message Queue . . . . .	289
Close a Named Semaphore . . . . .	171
close() . . . . .	
function definition . . . . .	103
closedir() . . . . .	
function definition . . . . .	85
COBOL . . . . .	354
Common-Usage C Language-Dependent System Support . . . . .	7
Common-Usage C Language-Dependent System Support . . . . .	7
Configurable Pathname Variables . . . . .	98
Configurable System Variables . . . . .	81, 83, 361
conformance 1, 1, 5-9, 25, 26, 28, 30-44, 46- 79, 81-83, 86, 87, 90-99, 103-154, 157, 160-189, 191-233, 235-277, 280-304 . . . . .	
definition of . . . . .	7
conformance requirements . . . . .	
definition of . . . . .	7
conforming implementation . . . . .	
definition of . . . . .	7
Conforming Implementation Options . . . . .	7
Conforming Test Results . . . . .	304
constant . . . . .	81, 82
Control Operations on Files . . . . .	106
controlling process . . . . .	
definition of . . . . .	24
controlling terminal . . . . .	
definition of . . . . .	24
Conventions . . . . .	8, 23
creat() . . . . .	
function definition . . . . .	87
Create a Per-Process Timer . . . . .	263
Create an Inter-Process Channel . . . . .	102
Create New File or Rewrite an Existing One . . . . .	87
Cross References . . . . .	9
CSMA/CD . . . . .	354
current working directory . . . . .	
change . . . . .	85
definition of . . . . .	24
D.C. . . . .	356
D_1 . . . . .	28, 48, 55, 58, 60, 70, 71, 73, 77, 90, 97, 99, 119, 125, 131, 143, 145, 148, 153, 157, 162, 164, 167, 172, 174, 178, 185, 188, 191, 195, 200, 204, 214, 217, 221, 225, 232, 235, 240, 243, 245, 249, 250, 252, 255, 260, 265, 268, 270, 276, 280, 282, 290, 292, 294, 297, 299, 301, 303-307, 309, 310, 312, 313, 316-321, 323-345, 347-352
D_10 . . . . .	170, 230, 329, 338
D_2 . . . . .	28, 44, 47, 55, 59, 61, 73, 97, 115, 119, 125, 143, 145, 162, 165, 168, 172, 178, 185, 191, 196, 200, 206, 214, 217, 221, 226, 238, 240, 243, 246, 253, 256, 265, 268, 273, 280, 283, 290, 294, 297, 301, 304, 306-310, 312-314, 317, 320-322, 325, 327- 337, 339, 340, 342-351
D_3 . . . . .	31, 56, 61, 74, 116, 120, 126, 162, 165, 168, 172, 191, 196, 200, 207, 214, 217, 221, 226, 238, 241, 246, 253, 257, 261, 266, 273, 281, 283, 304, 307-310, 313, 320-322, 327-329, 331-337, 339, 340, 342, 343, 346- 348
D_4 . . . . .	56, 62, 74, 116, 121, 126, 165, 168, 192, 197, 201, 207, 218, 221, 226, 241, 246, 257, 261, 267, 273, 285, 307-310, 313, 321, 322, 328, 332- 334, 336, 337, 339, 340, 343, 345, 346, 348
D_5 . . . . .	63, 75, 168, 193, 197, 201, 208, 221, 227, 241, 246, 257, 261, 273, 285, 310, 313, 328, 332-334, 336, 337, 339, 340, 343, 344, 346, 348
D_6 . . . . .	65, 169, 198, 202, 208, 222, 228, 257, 286, 311, 328, 332-334, 336, 337, 343
D_7 . . . . .	66, 169, 203, 209, 222, 228, 286, 311, 328, 333, 334, 336, 337, 348
D_8 . . . . .	67, 169, 203, 209, 229, 311, 328, 333, 334, 337, 349
D_9 . . . . .	67, 169, 229, 311, 328, 337
Data Definitions for Asynchronous Input and Output . . . . .	114
Data Definitions for Clocks and Timers . . . . .	255
Data Definitions for File Control Operations . . . . .	106
Data Definitions for Message Queues . . . . .	280
Data Interchange Format . . . . .	159, 327
definition of . . . . .	159
Data Structures . . . . .	280
Database Standards . . . . .	355
DCT . . . . .	354
Definitions . . . . .	24
Delay Process Execution . . . . .	80
delaytimer_max . . . . .	81, 273
Delete a Per-Process Timer . . . . .	267
Destroy an Unnamed Semaphore . . . . .	164
device . . . . .	
definition of . . . . .	24

Device- and Class- Specific Functions . . .	156	function definition . . . . .	54
definition of . . . . .	156	execle()	
Device- and Class-Specific Functions . . .	326	function definition . . . . .	54
DGA01 . . . . .	7	execlp()	
direct I/O		function definition . . . . .	54
definition of . . . . .	25	Execute a File . . . . .	54
Directories . . . . .	85	Execution Scheduling . . . . .	235, 338
directory . . . . .	24, 25, 27, 28, 32, 35-38, 40, 85,	definition of . . . . .	235
86, 88, 89, 99, 315, 359, 360, 362		execv()	
change current working . . . . .	85	function definition . . . . .	54
current working . . . . .	24	execve()	
definition of . . . . .	24	function definition . . . . .	54
empty . . . . .	25	f_dupd . . . . .	109
make . . . . .	88	f_dupfd . . . . .	107, 108
parent . . . . .	27	f_getfd . . . . .	107
remove . . . . .	89	f_getfl . . . . .	107
root . . . . .	28	f_setfd . . . . .	107
directory entry		f_setfl . . . . .	107, 108
definition of . . . . .	25	fchmod ()	
remove . . . . .	89	function definition . . . . .	92
directory entry [link]		fcntl()	
definition of . . . . .	25	function definition . . . . .	106
Directory Operations . . . . .	85	fdatasync()	
disarm (a timer)		function definition . . . . .	112
definition of . . . . .	25	fdopen()	
document . . . . .	1-3, 1, 2, 8, 9, 42, 355	function definition . . . . .	157
Documentation . . . . .	7	feature test macro	
dot		definition of . . . . .	25
definition of . . . . .	25	fifo . . . . .	25, 53, 55, 62, 70, 88, 116, 122, 123,
dot-dot		128-130, 137-140, 184, 185, 236-	
definition of . . . . .	25	238, 241, 252, 359	
drift rate (of a clock)		make . . . . .	88
definition of . . . . .	25	fifo scheduling policy	
DTE . . . . .	354	definition of . . . . .	236
dup()		FIFO special file	
function definition . . . . .	102	definition of . . . . .	25
dup2()		file	
function definition . . . . .	102	access . . . . .	92
Duplicate an Open File Descriptor . . . . .	102	block special . . . . .	24
effective group . . . . .	285	change modes . . . . .	92
effective group id . . . . .	25, 51, 94, 167, 285	change owner and group . . . . .	95
definition of . . . . .	25	character special . . . . .	24
effective user id . . . . .	12, 25, 76, 93-95, 167, 227,	close . . . . .	103
285		create or rewrite . . . . .	87
definition of . . . . .	25	definition of . . . . .	25
empty directory		execute . . . . .	54
definition of . . . . .	25	FIFO special . . . . .	25
empty string		link to . . . . .	87
definition of . . . . .	25	make FIFO special . . . . .	88
Epoch		open . . . . .	86
definition of . . . . .	25	read . . . . .	103
Examine and Change Blocked Signals . . . . .	70	regular . . . . .	28
Examine and Change Signal Action . . . . .	69	rename . . . . .	89
Examine Pending Signals . . . . .	71	write . . . . .	105
execl()		File Characteristics . . . . .	90

File Characteristics: Header and Data Structur ..... e89, 90	FORTTRAN ..... 2, 354
File Control ..... 106	fpathconf() function definition ..... 98
file description definition of ..... 25	fstat() function definition ..... 91
file descriptor 25, 95, 97, 98, 102, 103, 107- 109, 111-115, 122, 123, 125, 127- 129, 135-137, 139, 144-146, 149- 152, 154, 157, 204, 210, 225, 226, 229, 359	fsync() function definition ..... 110
duplicate ..... 102	ftruncate() function definition ..... 96
open stream ..... 157	FULL ..... 2, 293-295, 354
File Descriptor Deassignment ..... 103	functions
File Descriptor Manipulation ..... 102	_exit() ..... 57
file group class definition of ..... 25	access() ..... 92
file mode definition of ..... 25	aio_cancel() ..... 144
file offset definition of ..... 26	aio_error() ..... 141
file other class definition of ..... 26	aio_fsync() ..... 149
file owner class definition of ..... 26	aio_read() ..... 117
file permission bits definition of ..... 26	aio_return() ..... 142
file permissions ..... 26	aio_suspend() ..... 146
File Removal ..... 89	aio_write() ..... 123
file serial number definition of ..... 26	chdir() ..... 85
File Synchronization ..... 110	chmod() ..... 92
file system ..... 26, 28, 95, 98, 225, 282	chown() ..... 95
read-only ..... 28	close() ..... 103
filename definition of ..... 25	closedir() ..... 85
Files and Directories ..... 314	creat() ..... 87
definition of ..... 85	dup() ..... 102
first open (of a file) definition of ..... 26	dup2() ..... 102
for ... 1-3, 5, 1-3, 5-10, 13, 14, 23-83, 85-87, 90-99, 103-154, 157, 160-189, 191- 233, 235-277, 280-307, 309-333, 335, 336, 338-351, 353-355, 357, 359, 361, 362	execl() ..... 54
foreground ..... 26	execle() ..... 54
foreground process definition of ..... 26	execlp() ..... 54
foreground process group definition of ..... 26	execv() ..... 54
foreground process group ID definition of ..... 26	execve() ..... 54
fork() function definition ..... 52	execvp() ..... 54
Format of Directory Entries ..... 85	fchmod () ..... 92
	fcntl() ..... 106
	fdatasync() ..... 112
	fdopen() ..... 157
	fork() ..... 52
	fpathconf() ..... 98
	fstat() ..... 91
	fsync() ..... 110
	ftruncate() ..... 96
	getcwd() ..... 86
	kill() ..... 68
	link() ..... 88
	lio_listio() ..... 130
	lseek() ..... 109
	mkdir() ..... 88
	mkfifo() ..... 88
	mlock() ..... 198
	mlockall() ..... 194
	mmap() ..... 203
	mprotect() ..... 215

mq_getattr()	302	unlink()	89
mq_notify()	298	utime()	96
mq_open()	281	waitpid()	57
mq_receive()	295	write()	105
mq_send()	293	GA_macro_args	
mq_setattr()	300	General Assertion Definition	8
mq_unlink()	291	GA_macro_result_decl	
msync()	220	General Assertion Definition	8
munlock()	198	ga_queuedAndRegularSignals	
munmap()	213	General Assertion Definition	62
nanosleep()	275	GA_syncIODataIntegrityRead	
open()	86	General Assertion Definition	29
opendir()	85	GA_syncIODataIntegrityWbeforeR	
pause()	79	General Assertions	30
pipe()	102	GA_syncIOFileIntegrityRead	
read()	104	General Assertion Definition	30
readdir()	85	GA_syncIOFileIntegrityWrite	
rewinddir()	85	General Assertion Definition	31
rmdir()	89	ga_syncOpenWrite	
sched_get_priority_max()	250	General Assertion Definition	86
sched_get_priority_min()	250	GA_upperLowerNames	
sched_getscheduler()	248	General Assertion Definition	26
sched_rr_get_interval()	251	GA01	7
sched_setparam()	239	GA02	25
sched_setscheduler()	244	GA03	25
sched_yield()	249	GD_CommonC_diffs	
sem_close()	171	General Documentation Assertion	
sem_destroy()	164	definition	7
sem_getvalue()	187	General	5, 304
sem_init()	161	definition of	5
sem_open()	166	General Assertions	
sem_post()	183	ga_mqOpenMaxfd	281
sem_trywait()	175	ga_mqpctsOpenMaxfd	281
sem_unlink()	173	GA_portableFileNames	25
sem_wait()	175	GA_sigqueueValue	61
shm_open()	224	GA_syncIODataIntegrityRead	29
shm_unlink()	231	GA_syncIODataIntegrityWbeforeR	30
sigaction()	69	GA_syncIOFileIntegrityRead	30
sigaddset()	69	GA_syncIOFileIntegrityWrite	31
sigdelset()	69	ga_syncOpenWrite	86
sigfillset()	69	m_ga_sigPendingQueued	62
sigismember()	69	M_GA_upperLowerNames	26
sigpending	71	General Documentation Assertions	
sigprocmask()	70	GD_CommonC_diffs	7
sigqueue()	75	General File Creation	86
sigtimedwait()	72	General Terms	
sigwaitinfo()	72	definition of	24
sleep()	80	Get Configurable Pathname Variables	98
stat()	91	Get Configurable System Variables	81
sysconf()	81	Get File Status	91
timer_create()	263	Get Message Queue Attributes	302
timer_delete()	267	Get Scheduling Parameter Limits	250
timer_getoverrun()	269	Get Scheduling Parameters	242
timer_gettime()	269	Get Scheduling Policy	248
umask()	87	Get the Value of a Semaphore	187

Get Working Directory Pathname . . . . .	86	ISO 8649 . . . . .	353
getcwd()		ISO 8650 . . . . .	353
function definition . . . . .	86	ISO 8652 . . . . .	354
GKS . . . . .	354	ISO 8802-2 . . . . .	354
Graphics Standards . . . . .	354	ISO 8802-3 . . . . .	354
group		ISO 8822 . . . . .	354
change file . . . . .	95	ISO 8823 . . . . .	354
group ID		ISO 8831 . . . . .	354
definition of . . . . .	25, 26	ISO 8832 . . . . .	354
effective . . . . .	25	ISO 8859-1 . . . . .	355
real . . . . .	28	ISO 8907 . . . . .	355
saved set- . . . . .	28	ISO 9075 . . . . .	355
supplementary . . . . .	29	ISO 9127 . . . . .	355
High Resolution Sleep . . . . .	275	ISO/IEC 10646 . . . . .	355
Historical Documentation and Introductory		ISO/IEC 646 . . . . .	6, 9
Texts . . . . .	355	ISO/IEC 8073 . . . . .	353
IAP . . . . .	357	ISO/IEC 8802-4 . . . . .	354
IEEE . 1-4, 1-9, 24, 25, 27, 28, 32-52, 54, 57,		ISO/IEC 9592 . . . . .	355
61, 66, 68-71, 79-81, 84-92, 95, 96,		ISO/IEC 9899 . . . . .	6, 9
99, 100, 102-110, 157, 163, 222,		ISO/IEC 9945-2 . . . . .	355
235, 255, 256, 267, 280, 354, 355		IUT	
IEEE 802.5 . . . . .	354	Abbreviation . . . . .	9
IEEE Draft Std P2003-199x . . . . .	6	definition of . . . . .	9
IEEE Draft Std P2003-199X . . . . .	8	job control . . . . .	26, 49, 59
IEEE Std 100 . . . . .	355	definition of . . . . .	26, 59
IEEE Std 2003.1-1992 {4} . . . . .	6	Job Control Signals . . . . .	59
Implementation Conformance . . . . .	7	kill()	
implementation defined . . . . .	203	function definition . . . . .	68
Initialize an Unnamed Semaphore . . . . .	161	Language -Dependent Services for the C	
Initialize/Open a Named Semaphore . . . . .	165	Programming Language . . . . .	7
Input and Output . . . . .	103	Language Standards . . . . .	354
Input and Output Primitives . . . . .	102, 318	Language-Specific Services for the C	
int_max . . 162, 237-240, 242, 245, 247, 255,		Programming Language . . . . .	157
263, 275, 277		definition of . . . . .	157
inter-process channel . . . . .	102	Language-Specific Services for the C	
IPC . . . . .	102	Programming Language . . . . .	326
IPC (see inter-process channel) . . . . .	102	last close (of a file)	
IRV . . . . .	9	definition of . . . . .	26
abbreviation . . . . .	9	lifetime	
definition of . . . . .	9	session . . . . .	29
ISO 1539 . . . . .	354	link	
ISO 1989 . . . . .	354	definition of . . . . .	26
ISO 3166 . . . . .	355	link count	
ISO 639 . . . . .	355	definition of . . . . .	26
ISO 7498 . . . . .	353	Link to a File . . . . .	87
ISO 7942 . . . . .	354	link()	
ISO 7- . . . . .	6	function definition . . . . .	88
ISO 8072 . . . . .	353	lio_listio()	
ISO 8326 . . . . .	353	function definition . . . . .	130
ISO 8327 . . . . .	353	lio_nop	
ISO 8348 . . . . .	353	definition of . . . . .	117
ISO 8473 . . . . .	353	lio_nowait	
ISO 8571 . . . . .	353	definition of . . . . .	117
ISO 8632 . . . . .	355	lio_read	

definition of	117	Map Process Address to a Memory Object	
lio_wait			203
definition of	117	map_failed	
lio_write		definition of	209
definition of	117	mask	
List Directed i/o	130	file creation	87
Lock a Semaphore	175	MCL_CURRENT	
Lock/Unlock a Range of Process Address Spac	e198	definition of	195
Lock/Unlock the Address Space of a Process	194	MCL_FUTURE	
login		definition of	195
definition of	26	Memory Locking Functions	194
login name		Memory Management	191, 331
definition of	26	definition of	191
lseek()		Memory Mapping Functions	203
function definition	109	memory object	
M_GA_macro_args		definition of	26
test specification macro definition	7	Memory Object Synchronization	220
M_GA_macro_result_decl		Memory Protection Signals	60
test specification macro definition	8	memory-resident	
m_ga_mqOpenMaxfd ()		definition of	26
test specification macro definition	281	message	
m_ga_mqpctsOpenMaxfd ()		definition of	26
test specification macro definition	281	Message Passing	280
M_GA_portableFilenames		definition of	280
test specification macro definition	25	Message Passing Functions	281
m_ga_queuedAndRegularSignals		message queue	
test specification macro definition	62	definition of	26
m_ga_semOpenMaxfd		MIT	356
test specification macro definition	160	mkdir()	
m_ga_semPCTSOOpenMaxfd		function definition	88
test specification macro definition	160	mkfifo()	
M_GA_sigev_value()		function definition	88
test specification macro definition	61	mlock	
m_ga_sigPending		Reference Assertion definition	199
test specification macro definition	62	mlock()	
m_ga_sigPendingQueued		function definition	198
test specification macro definition	62	mlockall( )	
M_GA_sigqueueValue()		function definition	194
test specification macro definition	61	mmap	
M_GA_upperLowerNames		Reference Assertion definition	204
test specification macro definition	26	mmap()	
M_GD_CommonC_diffs		function definition	203
test specification macro definition	7	mmap_ENOTSUP	
macro		Reference Assertion definition	205
feature test	25	mmap_SIGBUS	
Macros	9	Reference Assertion definition	209
Make a Directory	88	mode	
Make a fifo Special File	88	definition of	26
Manifest Constants	117, 256	mprotect	
Manipulate Signal Sets	69	Reference Assertion definition	216
map		mprotect()	
definition of	26	function definition	215
		mprotect_ENOTSUP	
		Reference Assertion definition	217
		mq_attr	

C-language type definition . . . . .	280	C-language type definition . . . . .	280
mq_close		ms_async	
Reference Assertion definition . . . .	290	definition of . . . . .	221
mq_getattr		ms_invalidate	
Reference Assertion definition . . . .	302	definition of . . . . .	221
mq_getattr()		ms_sync	
function definition . . . . .	302	definition of . . . . .	221
mq_notify		msync	
Reference Assertion definition . . . .	298	Reference Assertion definition . . . .	220
mq_notify()		msync()	
function definition . . . . .	298	function definition . . . . .	220
mq_notify_ebusy		msync_einval	
Reference Assertion definition . . . .	300	Reference Assertion definition . . . .	222
mq_open		MUMPS . . . . .	354
Reference Assertion definition . . . .	282	munlock()	
mq_open()		function definition . . . . .	198
function definition . . . . .	281	munlockall	
mq_open_eaccess1		Reference Assertion definition . . . .	196
Reference Assertion definition . . . .	287	munlockall ()	
mq_open_eaccess2		function definition . . . . .	194
Reference Assertion definition . . . .	287	munlock	
mq_open_eexit		Reference Assertion definition . . . .	200
Reference Assertion definition . . . .	287	munmap	
mq_open_enoent		Reference Assertion definition . . . .	213
Reference Assertion definition . . . .	289	munmap()	
mq_open_max . . . . .	13, 45, 46, 81, 83, 288	function definition . . . . .	213
mq_prio_max . . . . .	45, 46, 81, 83, 294, 295	munmap_SIGSEV	
mq_receive		munmap_SIGSEV . . . . .	213
Reference Assertion definition . . . .	296	name	
mq_receive()		login . . . . .	26
function definition . . . . .	295	user . . . . .	32
mq_receive_eagain		name_max . . . .	13, 38, 45, 51, 170, 174, 230, 232, 289, 292
Reference Assertion definition . . . .	297	nanosleep()	
mq_send		function definition . . . . .	275
Reference Assertion definition . . . .	293	NDL . . . . .	355
mq_send()		Networking Standards . . . . .	353
function definition . . . . .	293	no_option . . . .	7-9, 29-34, 38, 39, 42-44, 47, 50, 52-58, 60-79, 86, 87, 90, 92-99, 103-106, 108-154, 157, 160-189, 191-233, 235-277, 280-303, 305- 352
mq_send_eagain		no_test . . . .	29-34, 42, 43, 50-58, 62-67, 76, 78, 79, 81-83, 86, 87, 90, 91, 93-95, 97, 103-106, 108-116, 118-130, 132- 140, 142, 145-154, 160-164, 166, 168, 170-172, 174, 176-189, 191- 193, 195-198, 200-203, 205-215, 219-224, 228, 230, 232, 235-242, 244-248, 250, 252, 255, 256, 259- 263, 265-268, 272-277, 280-292, 294, 295, 297-299, 301, 302, 305- 351
mq_send_einval			
Reference Assertion definition . . . .	295		
mq_send_emsgsize			
Reference Assertion definition . . . .	295		
mq_setattr			
Reference Assertion definition . . . .	300		
mq_setattr()			
function definition . . . . .	300		
mq_unlink			
Reference Assertion definition . . . .	291		
mq_unlink()			
function definition . . . . .	291		
mq_unlink_enoent			
Reference Assertion definition . . . .	292		
mqd_t			

- no\_test\_support . . . 29, 52-58, 62-67, 76, 78, 79, 81-83, 87, 90, 91, 93-95, 97, 108, 109, 112, 114-116, 118, 119, 121-130, 135-140, 142, 145-154, 160-164, 166, 170-172, 174, 176-189, 191-193, 195-198, 200-203, 205-215, 219-224, 230, 232, 236-242, 244-248, 255, 256, 259-263, 265-268, 272-275, 277, 281-292, 294, 295, 297-299, 301, 302, 306-311, 313-317, 319-336, 338-340, 342-351
- Normative References . . . . . 6
- Notify Process that a Message is Available on a Queue . . . . . 298
- null string
  - definition of . . . . . 25, 27
- o\_creat . . . . . 283
- Open a File . . . . . 86
- Open a Message Queue . . . . . 281
- Open a Shared Memory Object . . . . . 224
- Open a Stream on a File Descriptor . . . . . 157
- open file
  - definition of . . . . . 27
- open file description
  - definition of . . . . . 27
- open()
  - function definition . . . . . 86
- open\_max . 13, 45-47, 81, 83, 108, 109, 160, 163, 170, 281, 288
- opendir()
  - function definition . . . . . 85
- orphaned process group
  - definition of . . . . . 27
- Other C Language-Related Specifications . 7
- Other C Language-Related Specifications . 7
- Other Language-Related Specifications . . . 8
- Other Language-Related Specifications . . . 8
- Other Sources of Information . . . . . 357
- Other Standards . . . . . 355
- owner
  - change file . . . . . 95
- page
  - definition of . . . . . 27
- pagesize . . 11, 13, 47, 81, 83, 200, 202, 203, 207, 208, 212-215, 217, 219-221, 224
- parent directory
  - definition of . . . . . 27
- parent process
  - definition of . . . . . 27
- parent process ID
  - definition of . . . . . 27
- pass . . . 7, 8, 25, 26, 28, 30-44, 46-79, 81-83, 86, 87, 90-99, 103-154, 157, 160-189, 191-233, 235-277, 280-352
- path prefix
  - definition of . . . . . 27
- path\_max . . . . . 13, 45, 170, 230, 288
- pathname
  - absolute . . . . . 24
  - definition of . . . . . 27
  - get configurable values . . . . . 98
  - get working directory . . . . . 86
  - relative . . . . . 28
- pathname component
  - definition of . . . . . 27
- pause()
  - function definition . . . . . 79
- PCD.1b
  - abbreviation . . . . . 9
  - definition of . . . . . 9
- pcts Symbols and Values . . . . . 10
- PCTS Variables . . . . . 10
- pcts-ftuncate
  - PCTS variable definition . . . . . 16
- PCTS\_ . . 3, 5, 8, 10-21, 28-31, 52-58, 62-68, 70, 72-79, 87, 90, 91, 93-98, 103-105, 107-154, 160-189, 191-193, 195-233, 236-250, 252-254, 259-277, 281-303
- pcts\_aio\_cancel
  - pcts variable definition . . . . . 15
- pcts\_aio\_cancelable\_ops . . . . . 10, 144-146
  - PCTS variable definition . . . . . 10
- pcts\_aio\_error
  - PCTS variable definition . . . . . 15
- pcts\_aio\_fsync
  - PCTS variable definition . . . . . 15
- pcts\_aio\_max . . 10, 121, 127, 134, 135, 153
  - PCTS variable definition . . . . . 10
- pcts\_aio\_max . . . . . 134, 153
- pcts\_aio\_read
  - PCTS variable definition . . . . . 15
- pcts\_aio\_return
  - PCTS variable definition . . . . . 15
- pcts\_aio\_suspend
  - PCTS variable definition . . . . . 16
- pcts\_aio\_write
  - PCTS variable definition . . . . . 16
- pcts\_append\_write\_same\_order . . . . . 115
  - PCTS variable definition . . . . . 10
- pcts\_chmod\_sgid . . . . . 93, 94
  - PCTS variable definition . . . . . 10
- pcts\_chmod\_suid . . . . . 93
  - PCTS variable definition . . . . . 10
- pcts\_clock-gettime
  - PCTS variable definition . . . . . 16



pcts_clock_getres	PCTS variable definition . . . . .	16	PCTS variable definition . . . . .	12
pcts_clock_gettime	PCTS variable definition . . . . .	16	pcts_gap_sigqueue	PCTS variable definition . . . . .
pcts_defect_lockable_memory_limit_mlockall	PCTS variable definition . . . . .	11	pcts_gap_suid_fchmod	PCTS variable definition . . . . .
pcts_delaytimer_max . . . . .	273		pcts_get-priority_max	PCTS variable definition . . . . .
pcts_detect_aio_error_aiocbp . . . . .	144		pcts_get_priority_min	PCTS variable definition . . . . .
pcts_detect_enspc	PCTS variable definition . . . . .	10	pcts_gti_device . . . . .	13, 30, 104, 105, 116, 118-128, 130, 132, 133, 135, 137, 138, 140, 150
pcts_detect_eperm_clock_gettime	PCTS variable definition . . . . .	11		PCTS variable definition . . . . .
pcts_detect_invalid_flags_mmap	PCTS variable definition . . . . .	11	pcts_invalid_signal . . . . .	78
pcts_detect_lockable_memory_limit_mlock	PCTS variable definition . . . . .	11		PCTS variable definition . . . . .
pcts_detect_message_data_corruption . . . . .	11, 298		pcts_lio_listio	PCTS variable definition . . . . .
	PCTS variable definition . . . . .	11	pcts_map_fixed . . . . .	13, 207, 208, 211, 212
pcts_detect_no_ap . . . . .	198, 203			PCTS variable definition . . . . .
pcts_detect_not_multiple_of_pagesize . . . . .	11, 202, 203, 215, 219, 224		pcts_map_private . . . . .	13, 206, 211, 214, 218, 219, 221
	PCTS variable definition . . . . .	11		PCTS variable definition . . . . .
pcts_detect_not_multiple_of_pagesize . . . . .	215		pcts_mlock	PCTS variable definition . . . . .
pcts_einval_fchmod	PCTS variable definition . . . . .	11	pcts_mlockall	PCTS variable definition . . . . .
pcts_extend_on_ftruncate	PCTS variable definition . . . . .	12	pcts_mmap	PCTS variable definition . . . . .
pcts_fchmod	PCTS variable definition . . . . .	16	pcts_more_sa_siginfo_signals . . . . .	66
pcts_fdatasync	PCTS variable definition . . . . .	16		PCTS variable definition . . . . .
pcts_fsync	PCTS variable definition . . . . .	16	pcts_mprotect	PCTS variable definition . . . . .
pcts_gap_ . . . . .	58		pcts_mq_as_file_type . . . . .	90
pcts_gap_clock_gettime	PCTS variable definition . . . . .	12		PCTS variable definition . . . . .
pcts_gap_mlock . . . . .	214		pcts_mq_close	PCTS variable definition . . . . .
pcts_gap_mlockall . . . . .	214			PCTS variable definition . . . . .
	pcts variable definition . . . . .	12	PCTS_MQ_FILE_DESCRIPTOR . . . . .	281
pcts_gap_modes_fchmod	PCTS variable definition . . . . .	12	pcts_mq_getattr	PCTS variable definition . . . . .
pcts_gap_mq-open	PCTS variable definition . . . . .	12	pcts_mq_notify	PCTS variable definition . . . . .
pcts_gap_sched_setparam	PCTS variable definition . . . . .	12		PCTS variable definition . . . . .
pcts_gap_sched_setscheduler	PCTS variable definition . . . . .	13	pcts_mq_open	PCTS variable definition . . . . .
pcts_gap_sem_init	PCTS variable definition . . . . .	12		PCTS variable definition . . . . .
pcts_gap_sgid_fchmod			PCTS_MQ_OPEN_MAX . . . . .	13
				PCTS variable definition . . . . .
			pcts_mq_receive	PCTS variable definition . . . . .
				PCTS variable definition . . . . .
			pcts_mq_send	PCTS variable definition . . . . .
				PCTS variable definition . . . . .
			pcts_mq_setattr	PCTS variable definition . . . . .
				PCTS variable definition . . . . .
			pcts_mq_unlink	PCTS variable definition . . . . .
				PCTS variable definition . . . . .
			pcts_msync	

PCTS variable definition . . . . .	18	PCTS variable definition . . . . .	19
pcts_msync_storage		pcts_sched_rr_get_interval	
PCTS variable definition . . . . .	18	PCTS variable definition . . . . .	19
pcts_multiple_of_pagesize . . . . .	13, 202, 203, 207, 208, 212, 214, 215, 219, 224	pcts_sched_setparam	
PCTS variable definition . . . . .	13	PCTS variable definition . . . . .	19
pcts_munlock		pcts_sched_setscheduler	
PCTS variable definition . . . . .	18	PCTS variable definition . . . . .	19
pcts_munlockall		pcts_sched_yield	
PCTS variable definition . . . . .	18	PCTS variable definition . . . . .	19
pcts_munmap		pcts_sem_close	
PCTS variable definition . . . . .	18	PCTS variable definition . . . . .	19
pcts_name_max . . . . .	13, 170, 174, 230, 232, 289, 292	pcts_sem_destroy	
PCTS variable definition . . . . .	13	PCTS variable definition . . . . .	19
pcts_nanosleep		pcts_sem_ebusy . . . . .	165
PCTS variable definition . . . . .	18	PCTS variable definition . . . . .	14
pcts_no_sync_io_file . . . . .	13, 87, 108, 112, 114	pcts_sem_file_descriptors . . . . .	160
PCTS variable definition . . . . .	13	pcts_sem_getvalue	
pcts_open_max . . . . .	13, 109, 160, 163, 170, 281, 288	PCTS variable definition . . . . .	19
PCTS variable definition . . . . .	13	pcts_sem_init	
pcts_path_max . . . . .	13, 170, 230, 288	PCTS variable definition . . . . .	19
PCTS variable definition . . . . .	13	pcts_sem_invalid . . . . .	189
pcts_pipe_buf		PCTS variable definition . . . . .	14
PCTS variable definition . . . . .	14	pcts_sem_is_fd . . . . .	14, 90, 91, 163, 170
pcts_rap_clock_settime		PCTS variable definition . . . . .	14
pcts variable definition . . . . .	14	pcts_sem_nsems_max . . . . .	15, 57, 163, 164, 171
pcts_rap_mlock		PCTS variable definition . . . . .	15
PCTS variable definition . . . . .	14	pcts_sem_nsems_max . . . . .	171
pcts_rap_mlockall		pcts_sem_open	
PCTS variable definition . . . . .	14	PCTS variable definition . . . . .	20
pcts_rap_mq_open		pcts_sem_open_fd . . . . .	166
PCTS variable definition . . . . .	14	pcts_sem_post	
pcts_rap_sched_setparam		PCTS variable definition . . . . .	20
PCTS variable definition . . . . .	14	pcts_sem_trywait	
pcts_rap_sem_init		PCTS variable definition . . . . .	20
PCTS variable definition . . . . .	14	pcts_sem_unlink	
pcts_rap_sgid_chmod		PCTS variable definition . . . . .	20
PCTS variable definition . . . . .	14	pcts_sem_wait	
pcts_rap_sgid_fchmod		PCTS variable definition . . . . .	20
pcts variable definition . . . . .	14	pcts_shm_as_file_type . . . . .	15, 91
pcts_rap_sigqueue		PCTS variable definition . . . . .	15
PCTS variable definition . . . . .	14	pcts_shm_open	
pcts_read		PCTS variable definition . . . . .	20
PCTS variable definition . . . . .	18	pcts_shm_unlink	
pcts_rofs . . . . .	95	PCTS variable definition . . . . .	20
PCTS variable definition . . . . .	14	pcts_sigqueue	
pcts_sched_get_priority_max		PCTS variable definition . . . . .	20
PCTS variable definition . . . . .	18	pcts_sigqueue_max . . . . .	76
pcts_sched_get_priority_min		PCTS variable definition . . . . .	15
PCTS variable definition . . . . .	19	pcts_sigtimedwait	
pcts_sched_getparam		PCTS variable definition . . . . .	20
PCTS variable definition . . . . .	19	pcts_sigtimedwait_value . . . . .	75
pcts_sched_getscheduler		PCTS variable definition . . . . .	15
		pcts_sigwaitinfo	
		PCTS variable definition . . . . .	20
		pcts_timer_create	

PCTS variable definition . . . . .	20	definition of . . . . .	27
pcts_timer_delete		delay execution . . . . .	80
PCTS variable definition . . . . .	21	foreground . . . . .	26
pcts_timer_getoverrun		parent . . . . .	27
PCTS variable definition . . . . .	21	send signal to . . . . .	68
pcts_timer_gettime		suspend execution . . . . .	79
PCTS variable definition . . . . .	21	system . . . . .	31
pcts_timer_max		terminate . . . . .	57
PCTS variable definition . . . . .	15	wait for termination . . . . .	57
pcts_timer_settime		Process Creation . . . . .	52, 57
PCTS variable definition . . . . .	21	Process Environment . . . . .	81, 314
pcts_unsupported_signal . . . . .	78	definition of . . . . .	81
PCTS variable definition . . . . .	15	process group	
pcts_write		background . . . . .	24
PCTS variable definition . . . . .	21	definition of . . . . .	27
Per-Process Timers . . . . .	269	foreground . . . . .	26
permission		orphaned . . . . .	27
file . . . . .	26	process group ID	
persistence		definition of . . . . .	27
definition of . . . . .	27	foreground . . . . .	26
PHIGS . . . . .	355	process group leader	
Phrases . . . . .	9	definition of . . . . .	27
pipe . . . 11, 14, 27, 45, 59, 95, 102, 122, 123,		process group lifetime	
128-130, 137-140, 318, 360		definition of . . . . .	27
pipe()		process ID . . . . .	27, 240, 243, 245, 248
function definition . . . . .	102	definition of . . . . .	27
Pipes . . . . .	102	parent . . . . .	27
portable filename character set		process lifetime	
definition of . . . . .	27	definition of . . . . .	27
posix.1 . . . . .	41, 355	process list	
POSIX.1b . . . . .	6	definition of . . . . .	27
Abbreviation . . . . .	9	Process Primitives . . . . .	52, 306
definition of . . . . .	9	definition of . . . . .	52
POSIX.1tm		prot_exec	
Abbreviation . . . . .	9	definition of . . . . .	205
definition of . . . . .	9	prot_none	
POSIX.tm		definition of . . . . .	205
Abbreviation . . . . .	9	prot_read	
definition of . . . . .	9	definition of . . . . .	205
preallocation		prot_values	
definition of . . . . .	27	Reference Assertion definition . . . .	216
preempted process		prot_write	
definition of . . . . .	27	definition of . . . . .	216
priority		Queue a Signal to a Process . . . . .	75
definition of . . . . .	27	R_10 . . . . .	139, 324
priority-based scheduling		R_11 . . . . .	139, 324
definition of . . . . .	27	R_2 . . 29, 74, 95, 98, 114, 121, 127, 134, 143,	
privilege		146, 148, 153, 163, 165, 172, 174,	
definition of . . . . .	27	179, 186, 189, 191, 197, 201, 209,	
process		215, 223, 229, 232, 243, 246, 249,	
background . . . . .	24	250, 253, 261, 267, 269, 274, 277,	
child . . . . .	24	286, 290, 292, 294, 297, 299, 301,	
controlling . . . . .	24	303, 304, 313, 314, 317, 320-343,	
create inter-process channel . . . . .	102	345-352	

R_3	. 121, 129, 134, 143, 146, 148, 179, 186, 191, 197, 201, 209, 218, 223, 229, 246, 253, 261, 274, 277, 286, 292, 294, 297, 299, 321, 323, 325, 330, 331, 333, 334, 336, 337, 340, 341, 343, 346-351	mq_send_emsgsize	295
R_4	. 122, 130, 134, 149, 180, 192, 201, 210, 218, 253, 261, 274, 287, 294, 322, 323, 325, 330, 332, 334, 336, 342, 344, 346, 348, 350	mq_setattr	300
R_5	. 123, 130, 134, 192, 253, 261, 274, 287, 294, 322, 323, 332, 342, 344, 346, 348, 350	mq_unlink	291
R_6	. 123, 134, 193, 253, 262, 322, 323, 332, 342, 344	mq_unlink_enoent	292
R_7	..... 136, 323	msync	220
R_8	..... 137, 324	msync_einval	222
R_9	..... 137	munlock	201
Read from a File	..... 103	munlockall	196
read()		munmap	213
function definition	..... 103	munmap_SIGSEV	213
read-only file system		prot_values	205
definition of	..... 28	sched_get_priority_max	252
readdir()		sched_get_priority_min	252
function definition	..... 85	sched_getparam	243
real group ID		sched_getscheduler	248
definition of	..... 28	sched_rr_get_interval	252
real user ID		sched_setparam	239
definition of	..... 28	sched_setscheduler	244
Receive a Message From a Message Queue	..... 295	sched_yield	250
Reference Assertions		sem_close()	171
clock_getres	..... 259	sem_destroy()	164
clock_gettime	..... 259	sem_getvalue()	187
clock_settime	..... 259	sem_init()	161
mlock	..... 201	sem_open()	166
mlockall	..... 195	sem_post()	183
mmap	..... 205	sem_trywait()	175
mmap_ENOTSUP	..... 205	sem_unlink()	173
mmap_SIGBUS	..... 209	sem_wait()	175
mprotect	..... 216	shm_exist_err	228
mprotect_ENOTSUP	..... 217	shm_open	229
mq_close	..... 290	shm_unlink	231
mq_notify	..... 298	timer_create	264
mq_notify_ebusy	..... 300	timer_delete	268
mq_open	..... 282	timer_getoverrun	272
mq_open_eaccess1	..... 287	referenced shared memory object	
mq_open_eaccess2	..... 287	definition of	..... 28
mq_open_eexit	..... 287	region	
mq_open_enoent	..... 289	definition of	..... 28
mq_receive	..... 296	regular file	
mq_receive_eagain	..... 297	definition of	..... 28
mq_send	..... 293	Related Open Systems Standards	353
mq_send_eagain	..... 295	definition of	353
mq_send_einval	..... 295	relative pathname	
		definition of	..... 28
		Remove a Directory	89
		Remove a Message Queue	291
		Remove a Named Semaphore	173
		Remove a Shared Memory Object	231
		Remove Directory Entries	89
		Rename a File	89
		Reposition Read/Write File Offset	109
		Required Signals	59, 79
		Requirements	7
		Retrieve Error Status of Asynchronous i/o Operation	141

Retrieve Return Status of Asynchronous i/o Operation	142	Reference Assertion definition	244
rewinddir()	function definition	85	sched_setscheduler()
rmdir()	function definition	89	function definition
root directory	definition of	28	sched_yield
rtsig_max		81, 83	Reference Assertion definition
runnable process	definition of	28	sched_yield()
running process	definition of	28	function definition
s_isgid		93, 94	scheduling
s_isuid		93	definition of
s_typeismq		90	Scheduling Parameters
s_typeissem		90	235
s_typeisshm		90	Scheduling Policies
sa_siginfo		69	235
saved set-group-ID	definition of	28	scheduling policy
saved set-user-ID	definition of	28	definition of
sc_rtsig_max	limit definition	83	28
SCHED_FIFO	definition of	236, 252	Scope
sched_get_priority_max	Reference Assertion definition	252	5
sched_get_priority_max(),	function definition	250	seconds since the Epoch
sched_get_priority_min	Reference Assertion definition	252	definition of
sched_get_priority_min(),	function definition	250	28
sched_getparam	Reference Assertion definition	243	sem_close
sched_getscheduler	Reference Assertion definition	248	Reference Assertion definition
sched_getscheduler(),	function definition	248	172
SCHED_OTHER	definition of	236, 252	sem_close()
sched_param	C-language type definition	235	function definition
SCHED_RR	definition of	238, 252	171
sched_rr_get_interval()	function definition	251	sem_destroy
sched_setparam	Reference Assertion definition	239	Reference Assertion definition
sched_setparam(),	function definition	239	165
sched_setscheduler	Reference Assertion definition	244	sem_destroy()
	function definition	244	function definition
		249	164
		250	sem_getvalue
		249	Reference Assertion definition
		28	188
		28	sem_getvalue()
		5	function definition
		28	187
		28	sem_init
		172	Reference Assertion definition
		171	162
		165	sem_init()
		164	function definition
		188	161
		187	sem_nsems_max
		162	81
		166	sem_open
		166	Reference Assertion definition
		184	166
		183	sem_open()
		160	function definition
		175	166
		173	sem_post
		81	Reference Assertion definition
		175	184
		28	sem_post()
		160	function definition
		161	183
		28	sem_t
		160	C-language type definition
		175	160
		173	sem_trywait()
		81	function definition
		175	175
		28	sem_unlink()
		160	function definition
		161	173
		28	sem_value_max
		160	81
		161	sem_wait()
		28	function definition
		160	175
		161	semaphore
		28	definition of
		160	28
		161	Semaphore Characteristics
		161	160
		161	Semaphore Functions
		161	161

semaphore lock operation	
definition of	28
semaphore unlock operation:	
definition of	29
sen_nsems_max	163, 171
Send a Message to a Message Queue	293
Send a Signal to a Process	68
serial number	
file	26
session	353
definition of	29
session leader	
definition of	29
session lifetime	
definition of	29
Set File Access and Modification Times	96
Set File Creation Mask	87
Set Message Queue Attributes	300
Set Scheduling Parameters	239
Set Scheduling Policy and Scheduling Parameters	244
SETUP	114
Shared Memory Functions	224
shared memory object	
definition of	29
shm_exist_err	
Reference Assertion definition	228
shm_open()	
function definition	224
Reference Assertion definition	225
shm_unlink	
Reference Assertion definition	231
shm_unlink()	
function definition	231
si_user	
definition of	65
sigabrt	
definition of	59
sigaction()	
function definition	69
sigaddset()	
function definition	69
sigalrm	
definition of	59
sigbus	
definition of	60
sigchld	
definition of	60
sigcont	
definition of	60
sigdelset()	
definition of	69
sigemptyset()	
function definition	69
sigevent	
C-language type definition	60
sigfillset()	
function definition	69
sigfpe	
definition of	59
sigfpe	
definition of	59
sigill	
definition of	59
siginfo_t	
C-language type definition	64
sigint	
definition of	59
sigismember()	
function definition	69
sigkill	
definition of	59
signal	
concepts	59
definition of	29
examine and change action	69
examine and change blocked	70
examine pending	71
manipulate sets	69
mask	70
send to process	68
wait	71
Signal Actions	63
Signal Concepts	59
Signal Effects on Other Functions	68
Signal Generation and Delivery	60
Signal Names	59
Signals	59
sigpending()	
function definition	71
sigpipe	
definition of	59
sigprocmask()	
function definition	70
sigqueue()	
function definition	75
sigqueue_max	76
sigqueue_priv()	
test specification macro definition	76
sigquit	
definition of	59
sigrtmax	
definition of	60
sigrtmin	
definition of	60
sigsegv	
definition of	59
sigstop	
definition of	60
sigsuspend()	

function definition . . . . .	71	synchronous I/O operation	
sigterm		definition of . . . . .	31
definition of . . . . .	59	Synchronously Accept a Signal . . . . .	71
sigtimedwait()		system	
function definition . . . . .	72	definition of . . . . .	31
sigtstp		system crash	
definition of . . . . .	60	definition of . . . . .	31
sigttin		System Databases . . . . .	158, 326
definition of . . . . .	60	definition of . . . . .	158
sigttou		system process	
definition of . . . . .	60	definition of . . . . .	31
sigusr1		system reboot	
definition of . . . . .	59	definition of . . . . .	31
sigusr2		System V . . . . .	355
definition of . . . . .	59	terminal	
sigwaitinfo()		controlling . . . . .	24
function definition . . . . .	72	definition of . . . . .	31
slash		Terminate a Process . . . . .	57
definition of . . . . .	29	Terminology . . . . .	24
sleep()		Terminology and General Requirements	
function definition . . . . .	80	. . . . .	304
Special File Creation . . . . .	88	definition of . . . . .	23
Special Symbol clk_tck . . . . .	84	Terms . . . . .	24
SQL . . . . .	355	Test Methods . . . . .	8
standards		test specification macros	
database . . . . .	355	M_GA_macro_args . . . . .	7
graphics . . . . .	354	M_GA_macro_result_decl() . . . . .	8
language . . . . .	354	m_ga_mqOpenMaxfd () . . . . .	281
networking . . . . .	353	m_ga_mqpctsOpenMaxfd () . . . . .	281
open systems . . . . .	353	M_GA_portableFileNames . . . . .	25
stat()		m_ga_queuedAndRegularSignals . . . . .	62
function definition . . . . .	91	M_GA_semOpenMaxFD() . . . . .	160
stream		M_GA_semPCTSOpenMaxFD() . . . . .	160
open . . . . .	157	M_GA_sigev_value () . . . . .	61
successfully transferred		m_ga_sigPending . . . . .	62
definition of . . . . .	29	m_ga_sigPendingQueued . . . . .	62
supplementary group		M_GA_sigqueueValue() . . . . .	61
definition of . . . . .	29	M_GA_stdC_proto_decl . . . . .	9
supplementary group ID		M_GA_upperLowerNames . . . . .	26
definition of . . . . .	29	M_GD_CommonC_diffs . . . . .	7
supplementary groups . . . . .	94	sigqueue_priv() . . . . .	76
Suspend Process Execution . . . . .	79	Time Value Specification Structures . . . . .	255
Synchronization . . . . .	160, 327	timer	
definition of . . . . .	160	definition of . . . . .	31
Synchronize the Data of a File . . . . .	112	Timer Event Notification Control Block	
Synchronize the State of a File . . . . .	110	. . . . .	256
synchronized I/O completion		Timer Operations . . . . .	79
definition of . . . . .	29	timer overrun	
synchronized I/O data integrity completion		definition of . . . . .	31
definition of . . . . .	29	timer_abstime	
synchronized I/O file integrity completion		definition of . . . . .	256
definition of . . . . .	30	timer_create	
synchronized I/O operation		Reference Assertion definition . . . . .	264
definition of . . . . .	31	timer_create()	

function definition . . . . .	263
timer_delete	
Reference Assertion definition . . . .	268
timer_delete()	
function definition . . . . .	267
timer_getoverrun	
Reference Assertion definition . . . .	272
timer_getoverrun()	
function definition . . . . .	269
timer_gettime()	
function definition . . . . .	269
timer_max . . . . .	81
timer_settime()	
function definition . . . . .	269
TR 10000 . . . . .	355
TRUE . . . . .	10
Truncate a File to a Specified Length . . .	96
TSG-1 Final Report . . . . .	357
TSG-1 . . . . .	357
Type Definitions . . . . .	256
Types of Conformance . . . . .	7
uint_max . . . . .	295
umask()	
function definition . . . . .	87
undefined . . . . .	56, 60
UNIX . . . . .	355, 356
unlink()	
function definition . . . . .	89
Unlock a Semaphore . . . . .	183
Unmap Previously Mapped Addresses . .	213
unspecified . . . . .	187, 188
USENIX . . . . .	356
user ID	
definition of . . . . .	32
effective . . . . .	25
real . . . . .	28
saved set- . . . . .	28
user name	
definition of . . . . .	32
utime()	
function definition . . . . .	96
VAX-11 . . . . .	356
Version 7 . . . . .	356
Wait for a Signal . . . . .	71
Wait for Asynchronous i/o Request . . . .	146
Wait for Process Termination . . . . .	57
waitpid()	
function definition . . . . .	57
working directory	
change . . . . .	85
Write to a File . . . . .	105
write()	
function definition . . . . .	105
X.25 . . . . .	354
X/Open . . . . .	357

Yield Processor . . . . .	249
---------------------------	-----