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. lb-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

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*

Editor's Notes

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

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

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

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

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

Please report typographical errors to:

Barry Hedquist
Perennial, Inc.
15810 Miradero Ave, Suite 200.
San Jose, CA 95127
TEL: +1 (408) 347-7800
FAX: +1 (408) 347-7803
Email: beh@peren.com

(Electronic mail is preferred.)

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

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

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

P2003.1b Change History

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

Draft 3	First complete draft of all of POSIX.1b {3} assertions. *PCTS_* variables are defined. Subclause 1.4 was added, but is incomplete.
Draft 4	First Balloted Draft, revision, and more complete version of Draft 3.
Draft 5	First Recirculation Ballot. Contained ballot resolutions for Draft 4.
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

Contents

TABLES

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)

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 this 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.

Organization of This Standard

This document is organized into five parts:

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

(2) Conventions and definitions (Section 2)

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

(4) Conforming test results (Annex A)

(5) Bibliography (Annex B)

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

How to Read This Standard

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

Related Standards Activities

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

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

(1) Language-independent service descriptions of this standard

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

(3) Shell and utility facilities

(4) Verification testing methods

(5) Multi-threaded process facilities

(6) Secure/Trusted system considerations

(7) Network interface facilities

(8) System administration

(9) Graphical user interfaces

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

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

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

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

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

---

1)

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

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

Portable Applications Standards Committee

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

2003 Working Group Officials

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

Technical Reviewers

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

2003.1b Working Group

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

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

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

1.1 Scope

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

This standard is intended for use by

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

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

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

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

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

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

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

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

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

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

1.2 Normative References

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

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

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

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

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

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

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

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

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

1.3.1.2 Documentation

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

1.3.1.3 Conforming Implementation Options

There are no requirements for conforming implementations in this subclause.

1.3.2 Application Conformance

There are no requirements for conforming implementations in this subclause.

1.3.3 Language -Dependent Services for the C Programming Language

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

1.3.3.1 Types of Conformance

There are no requirements for conforming implementations in this subclause.

1.3.3.2 C Standard Language-Dependent System Support

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

1.3.3.3 Common-Usage C Language-Dependent System Support

(IEEE Std 2003.1-1992 {4} DGA01
UNUSED

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

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

1.3.4 Other C Language-Related Specifications

(IEEE Std 2003.1-1992 {4} GA01
UNUSED

M_GA_macro_args (function; header1; header2; header3; header4) =
IF the interface *function* () is defined as a macro **THEN**
SETUP: The headers `<header1>, <header2>, <header3>,` and `<header 4>` are included.

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

ELSE *NO_OPTION*

GA_macro_args

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

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

Conformance for conformance: PASS, NO_OPTION

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

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

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

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

ELSE *NO_OPTION*

GA_macro_result_decl

FOR: All functions implemented as macros

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

Conformance for conformance: PASS, NO_ OPTION

1.3.5 Other Language-Related Specifications

There are no requirements for conforming implementation in this subclause.

1.4 Test Methods

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

1.4.1 Conventions

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

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

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

(IEEE Std 2003.1-1992 {4} 03

UNUSED

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

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

1.4.1.1 Phrases

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

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

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

There are no requirements for conforming implementations in this subclause.

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

1.4.1.2 Macros

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

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

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

1.4.1.3 Cross References

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

1.4.2 Abbreviations

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

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

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

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

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

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

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

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

1.4.3 PCTS Variables

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

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

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

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

Table 1-1 – PCTS Symbols and Values

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

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

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

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

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

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

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

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

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

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

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

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

THIS PAGE IS INTENTIONALLY BLANK.

Section 2: Terminology and General Requirements

2.1 Conventions

This standard uses the following typographic conventions:

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

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

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

-- Global external variable names

-- General Assertion and General Documentation Assertion references

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

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

- For C language data types and function names within function Synopsis subclauses
- To illustrate examples of system input or output where exact usage is depicted
- For references to utility names and C language headers

- Symbolic constants returned by many functions as error numbers are represented as:

  [ERRNO]

- Symbolic Constants or limits defined in certain headers are represented as:

  [LIMIT]

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

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

The conventions listed previously are for ease of reading only. Editorial inconsistencies in the use of typography6 are unintentional and have no normative meaning in this standard.

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

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

2.2 Definitions

2.2.1 Terminology

There are no requirements for conforming implementation in this subclause.

2.2.2 General Terms

There are no requirements for conforming implementations in this subclause.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2.2.23 direct I/O: There are no requirements for conforming implementations in this subclause.

2.2.2.24 disarm (a timer): There are no requirements for conforming implementation of this subclause.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(IEEE Std 2003.1-1992 {4} GA03
UNUSED

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2.2.105 scheduling policy:

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

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

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

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

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

(5) In other scheduling–policy-defined circumstances

Conformance for definitions: PASS

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

Conformance for definitions: PASS

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

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

2.2.2.108 semaphore lock operation:

R_1 FOR: *sem_init()* and *sem_open()*
IF *PCTS_sem_wait* ***THEN***
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function* ().
TEST: When a call to *sem_wait (sem)* complete successfully, the interface returns a value of 0 and the semaphore designated by *sem* is locked by the semaphore lock operation.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion sem_wait in §11.2.7.2

2.2.2.109 semaphore unlock operation:

R_2 FOR: *sem_init()* and *sem_open()*
IF *PCTS_sem_post* ***THEN***
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function* ().
TEST: A successful call to *sem_post*() unlocks the semaphore referenced by *sem* by performing the semaphore unlock operation on that semaphore, and returns the value zero.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion sem_post in §11.2.7.2

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

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

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

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

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

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

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

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

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

2.2.2.119 synchronized I/O data integrity completion:

GA_syncIODataIntegrityRead

FOR: *read(), aio_read(),* and *lio_listio()*
IF *PCTS_function* and {POSIX_SYNCH_IO} **THEN**
SETUP: Open a file by calling *open*() with O_RSYNC and O_DSYNC set in the *oflag* parameter.
TEST: At the time that the synchronized read operation initiated by calling *function*() occurs, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data.
TR: Test for regular files.
ELSE *NO_OPTION*
Conformance for definitions: PASS, NO_TEST, NO_OPTION

GA_syncIODataIntegrityWbeforeR

FOR: *write(), aio_write(),* and *lio_liosio()*
IF *PCTS_function* and {POSIX_SYNCH_IO} **THEN**
SETUP: Open a file by calling *open()* with O_DSYNC set in the *oflag* parameter.
TEST: A write operation initiated by calling *function*() either completes by transferring an image of the data to the physical medium containing the file or, if unsuccessful, by diagnosing and returning an indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.
ELSE *NO_OPTION*
Conformance for definitions: PASS, NO_TEST, NO_OPTION

GA_syncIODataIntegrityWrite

FOR: *write*(), *aio_write*(), and *lio_listio*()
IF *PCTS_funtion and* {POSIX_SYNCH_IO} **THEN**
SETUP: Open a file by calling *open*() with O_DSYNC set in the *oflag* parameter.
TEST: A write operation initiated by calling *function*() either completes by transferring an image of the date to the physical medium containing the file or, if unsuccessful, by diagnosing and returning and indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.
ELSE *NO_OPTION*
Conformance for definitions: PASS, NO_TEST, NO_OPTION

2.2.2.120 synchronized I/O file integrity completion:

GA_syncIOFileIntegrityRead

FOR: *read(), aio_read(),* and *lio_listio()*
IF *PCTS_function* and {POSIX_SYNCH_IO} **THEN**
SETUP Open a file by calling *open()* with O_RSYNC and O_SYNC set in the *oflag* parameter.
TEST: At the time that the synchronized read operation initiated by calling *function()* occurs, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

6. File mode.
2. File serial number
3. ID of device containing this file.
4. Number of links.
5. User ID of the owner of the file.

6. Group ID of the group of the file.

7. The file size in bytes.

8. Time of last access.

9. Time of last data modification.

10. Time of last file status change.

TR: Test for regular files.

ELSE *NO_OPTION*

Conformance for definitions: PASS, NO_TEST, NO_OPTION

GA_syncIOFileIntegrityWrite

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

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

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

1. File mode.

2. File serial number.

3. ID of device containing this file.

4. Number of links.

5. User ID of the owner of the file.

6. Group ID of the group of the file.

7. The file size in bytes.

8. Time of last access.

9. Time of last data modification.

10. Time of last file status change.

TR: Test for regular files.

ELSE *NO_OPTION*

Conformance for definitions: PASS, NO_TEST, NO_OPTION

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

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

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

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

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

2.2.2.126 system reboot:

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

Conformance for definitions: PASS

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

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

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

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

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

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

2.2.3 Abbreviations

There are no requirements for conforming implementations in this subclause.

See subclause 1.4.2 for abbreviations related to this standard.

2.3 General Concepts

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

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

UNUSED

M_GA_AP_overrideFileAccess*(function()) =*

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

SETUP: The process has appropriate privileges for file access.

TEST: A call to the interface *function*() that needs read, write, or search access to the *pathname* argument is granted access to the file when access would otherwise be denied..

ELSE *NO_OPTION*

GA_AP_overrideFileAccess

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

M_GA_AP_overrideFileAccess*(function())*

Conformance for General Concepts; PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4} GA05

UNUSED

M_GA_AP_overrideExecAccess*(function()) =*

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

SETUP: The process has appropriate privilege for the file access and execute permission is granted to at least one user of the file.

TEST: A call to the interface *function*() that needs execute permission to the *path* or *file* argument is granted execute access to the file when access would otherwise be denied.

ELSE *NO_OPTION*

GA_AP_overrideExecAccess

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

M_GA_AP_overrideExecAccess*(function())*

Conformance for General Concepts: PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4} GA06

UNUSED

M_GA_AP_classAccess*(function()) =*

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

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

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

ELSE *NO_OPTION*

GA_AP_classAccess

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

M_GA_AP_ClassAccess*(function())*

Conformance for General Concepts: PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4} GA07

UNUSED

M_GA_AdditionalAccessControl*()) =*

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

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

ELSE *NO_OPTION*

GA_AdditionalAccessControl

MP_GA_AdditionalAccessControl()

Conformance for General Concepts: PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4} GA08

UNUSED

MP_GA_AlternateAccessControl() =

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

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

ELSE *NO_OPTION*

GA_AlternateAccessControl

M_GA_AlternateAccessControl()

Conformance for General Concepts, PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4}) GA09

UNUSED

(IEEE Std 2003.1-1992 {4}) GA10

UNUSED

M_GA_AltAccessEnable () =

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

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

ELSE *NO_OPTION*

GA_AltAccessEnable

M_GA_AltAccessEnable()

Conformance for General Concepts: PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4}) GA11

UNUSED

M_GA_AltAccessDisable() =

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

TEST: The alternate file access control mechanisms will be disabled for a file after the file permission bits are changed for that file with *chmod*().

ELSE *NO_OPTION*

GA_AltAccessDisable

M_GA_AltAccessDisable()

Conformance for General Concepts: PASS, NO_TEST, NO_OPTION

(IEEE Std 2003.1-1992 {4}) D03

UNUSED

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

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

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

UNUSED

(IEEE Std 2003.1-1992 {4}) GA12

UNUSED

***M_GA_StatTimeUpdate** (function()) =*

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

GA_StatTimeUpdate

FOR: *stat()* and *fstat().*

***M_GA_StatTimeUpdate**(function())*

Conformance for General Concepts: PASS

***M_GA_NoOpenTimeUpdate**(function()) =*

TEST: All fields that are marked for update are updated when the file is no longer open by any process.

GA_NoOpenTimeUpdate

FOR: *close()* and *fclose*().

***M_GA_NoOpenTimeUpdate** (function())*

Conformance for General Concepts: PASS

NOTE: This assertion is missing in 2003.1

(IEEE Std 2003.1-1992 {4}_GA13

UNUSED

M_GA_NoROFSTimeUpdate*(function()) =*

TEST: Time-related field updates are not done for files on read-only file systems.

GA_NoROFSTimeUpdate

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

M_GA_noROFSTimeUpdate*(function())*

Conformance for General Concepts: PASS

2.3.6 pathname resolution:

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

(IEEE Std 2003.1-1992 {4}) GA14

UNUSED

M_GA_PRDot*(function()) =*

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

GA_PRDot

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

M_GA_PRDot*(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}_ GA15

UNUSED

M_GA_PRSlash*(function()) =*

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

GA_PRSlash

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

M_GA_PRSlash*(function())*

Conformance for General Concepts: pass

(IEEE Std 2003.1-1992 {4}) GA16

UNUSED

M_GA_PR3Slash*(function()) =*

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

GA_PR3Slash

FOR: *access(), chdir(), chmod(), chown(), creat(), execl(), execle(), execv(), execve(), execlp(), execvp(), fopen(), fre0-en(), open(), opendir(), link() existing, link() new, mkdir(), mkfifo(), pathconf(), remove(), rename() new, rename() old, rmdir(), stat(), unlink(),* and *utime().*

M_GA_PR3Slash*l**function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}) GA17

UNUSED

M_GA_PRSlashsPath*(function()) =*

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

GA_PRSlashsPath

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

M_GA_PRSlashsPath*(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}) GA18

UNUSED

M_GA_PRDotDot*(function()) =*

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

GA_PRDotDot

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

M_GA_PRDotDot*(function())*

Conformance for General Concepts; PASS

(IEEE Std 2003.1-1992 {4}) GA19

UNUSED

M_GA_PRRelativeSlash*(function()) =*

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

GA_PRRelativeSlash

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

M_GA_PRRelativeSlash*(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4} GA20

UNUSED

M_GA_PRRelativeSlashSlash*(function()) =*

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

GA_PRRelativeSlashSlash

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

M_GA_PRRelativeSlashSlash*(function())*

Conformance for General Concepts: PASS

M_GA_PRRenameRelativeSlashSlash*(function()) =*

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

GA_PRRenameRelativeSlashSlash

FOR: *rename()new*

M_GA_PRRenameRelativeSlashSlash*(function()*)

Conformance for general Concepts: PASS

(IEEE Std 2003.1-1992 {4})GA21

UNUSED

M_GA_PRRelativeCWD*(function()) =*

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

GA_PRRelativeCWD

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

M_GA_PRRelativeCWD*(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}_ GA22

UNUSED

M_GA_PRRelativeDotCWD*(function()) =*

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

GA_PRRelativeDotCWD

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

M_GA_PRRelativeDotCWD*(function())*

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4} GA23

UNUSED

M_GA_PRRelativeDotDotCWD*(function()) =*

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

GA_PRRelativeDotDotCWD

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

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

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}) GA24

UNUSED

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

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

GA_PRRelativeSlashSlashCWD

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

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

Conformance for General Concepts: PASS

(IEEE Std 2003.1-1992 {4}) GA25

UNUSED

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

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

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

ELSE *NO_OPTION*

GA_PRNoTrunc

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

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

Conformance for General Concepts: PASS, NO_OPTION

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

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

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

ELSE *NO_OPTION*

GA_PRNoTruncError

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

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

Conformance for General Concepts: PASS, NO_OPTION

2.4 Error Numbers

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

UNUSED

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

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

Conformance for Error numbers: PASS

(IEEE Std 2003.1-1992 {4} DGA02

UNUSED

M_GD_OptionalErrors*(function()) =*

IF the IUT supports the detection of an optional error condition **THEN**

TEST: The PCD.1b contains the details of the optional error conditions detected in the subclause of the PCD.1b where the error values of the interface *function*() are described.

ELSE *NO_OPTION*

GD_OptionalErrors

FOR: *access(), chown(), chosedir(), execl(), execle(), execv(), execve(), execlp(), execvp(), fcntl(), fork(), fpathconf(), getcwd(), opendir(), pathconf(), readdir(), sigaddset(), sigdelset(),* and *(sigismember().*

M_GD_OptionalErrors*(function())*

Conformance for Error Numbers: PASS, NO_OPTION

(IEEE Std 2003.1-1992 {4}) GA26

UNUSED

M_GA_OptionalErrorsUndetected*(function()) =*

IF the IUT does not supports the detection of an optional error condition **THEN**

TEST: The action specified by a call to the interface *function*() that would otherwise generate an optional error condition will succeed.

ELSE *NO_OPTION*

GA_OptionalErrorsUndetected

FOR: *access(), chown(), chosedir(), execl(), execle(), execv(), execve(), execlp(), execvp(), fcntl(), fork(), fpathconf(), getcwd(), opendir(), pathconf(), readdir(), sigaddset(), sigdelset(),* and *(sigismember().*

M_GA_OptionalErrorsUndetected*(function())*

Conformance for Error Numbers: PASS, NO_OPTION

2.5 Primitive System Data Types

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

2.6 Environment Description

(IEEE Std 2003.1-1992 {4}) GA27

UNUSED

M_GA_ExecNOSlash () =

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

GA_ExecNoSlash

FOR: *execlp()* and *execvp().*

M_GA_ExecNoSlash()

Conformance for Environment Description: PASS

(IEEE Std 2003.1-1992 {4}) GA28

UNUSED

M_GA_ExecColon() =

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

GA_ExecColon

FOR: *execlp()* and *execvp().*

M_GA_ExecColon()

Conformance for Environment Description: PASS

(IEEE Std 2003.1-1992 {4}) GA29

UNUSED

M_GA_ExecInsertSlash() =

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

GA_ExecInsertSlash

FOR: *execlp()* and *execvp().*

M_GA_ExecInsertSlash()

Conformance for environment Description: PASS

(IEEE Std 2003.1-1992 {4} GA30

UNUSED

M_GA_execTwoColons() =

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

GA_ExecTwoColons

FOR: *execlp()* and *execvp().*

M_GA_ExecTwoColons()

Conformance for Environment Description: PASS

(IEEE Std 2003.1-1992 {4}) GA31

UNUSED

M_GA_ExecInitialColon() =

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

GA_ExecInitialColon

FOR: *execlp()* and *execvp().*

M_GA_ExecInitialColon()

Conformance for Environment Description: PASS

(IEEE Std 2003.1-1992 {4}) GA32

UNUSED

M_GA_ExecTrailingColon() =

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

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

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

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

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

(IEEE Std 2003.1-1992 {4}) GA34
UNUSED

M_GA_EnvironCaseSensitive(function) =
TEST: The interface function retains the unique identities of upper-and lowercase letters in the environment and does not fold them together.

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

(IEEE Std 2003.1-1992 {4}) GA35
UNUSED

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

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

2.7 C Language Definitions

2.7.1 Symbols From the C Standard

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

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

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

2.7.2 POSIX.1 Symbols

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

5 **FOR:** Headers <aio.h>, <dirent.h>, <fcntl.h>, <grp.h>, <limits.h>, <locale.h>, <mqueue.h>, <pwd.h>, <sched.h>, <semaphore.h>, <signal.h>, <sys/mman.h>, <sys/stat.h>, <sys/times.h>, <>sys/wait.h>, <termios.h>, <time.h>, <unistd.h> and <utime.h>
IF the feature test macro_POSIX_C_SOURCE is defined to have at least the value 199309L **THEN**
TEST: All symbols required by IEEE Std 1003.1b_1993 to appear when a header is included shall be made visible when the _POSIX_C_SOURCE feature test macro is defined.
NOTE: The assertion test would require an unreasonable amount of time or resources on most implementations.
ELSE *NO_OPTION*
Conformance for C Language Definitions: PASS, NO_TEST, NO_OPTION

6 **FOR:** <aio.h>, <dirent.h>, <fcntl.h>, <grp.h>, <limits.h>, <locale.h>, <mqueue.h>, <pwd.h>, <sched.h>, <semaphore.h>, <signal.h>, <sys/mman.h>, <sys/stat.h>, <sys/times.h>, <>sys/wait.h>, <termios.h>, <time.h>, <unistd.h> and <utime.h>
TEST: When a header is included, additional symbols not required or explicitly permitted by IEEE Std 1003.1b-1993 or the C Standard {2} to be in that header shall not be made visible, except when enabled by another feature test macro or by having defined _POSIX_C_SOURCE with a value larger than 199309L
NOTE: The assertion test would require an unreasonable amount of time or resources on most implementations.
Conformance for C Language Definitions: PASS, NO_TEST

(IEEE Std 2003.1-1992 {4})C01
UNUSED

D-1 **IF** the IUT supports feature test macros in addition to _POSIX_C_SOURCE **THEN**
TEST: The PCD.1b either documents the additional feature test macros in subclause 2.7.2 or it does not document them at all.
ELSE *NO_OPTION*
Conformance for C Language Definitions: PASS, NO_OPTION

2.7.2.1 C Standard Language-Dependent Support

7 **SETUP:** A program does not use any feature test macros.
TEST: The IUT makes visible only those identifiers specified as reserved identifiers in the C Standard {2}.
NOTE: The assertion test requires setup procedures that involve an unreasonable amount of effort by the user of a test method.
Conformance for C Language Definitions: PASS, NO_TEST

8 **FOR:** Each feature test macro present.
TEST: The IUT makes visible only those identifiers specified by that feature test macro and those of the C Standard {2} when a header is included.
NOTE: The assertion test requires setup procedures that involve an unreasonable amount of effort by the user of a test method.

Conformance for C Language Definitions: PASS, NO_TEST

2.7.2.2 Common-Usage-Dependent Support

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

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

2.7.3 Headers and Function Prototypes

(IEEE Std 2003.1-1992 {4} GA36
UNUSED

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

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

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

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

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

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

M_GA_setjmpDecl( )=

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

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

ELSE *NO_OPTION*

GA_setjmpDecl

FOR: *setjmp* ().

M_GA_*setjmpDecl()*

Conformance for C Language Definitions: PASS, NO_OPTION

M_GA_sigsetjmpDecl()=

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

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

ELSE *NO_OPTION*

GA_sigsetjmpDecl

FOR: *sigsetjmp*().

M_GA_sigsetjmpDecl()

Conformance for C Language Definitions: PASS, NO_OPTION

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

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

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

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

ELSE *NO_OPTION*

GA_macro_args

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

Conformance for C Language Definitions: PASS, NO_OPTION

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

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

ELSE *NO_OPTION*

Conformance for C Language Definitions: PASS, NO_OPTION

2.8 Numerical Limits

There are no requirements for conforming implementations in this subclause.

2.8.1 C Language Limits

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

2.8.2 Minimum Values

(IEEE Std 2003.1-1992 {4}) 02

UNUSED

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

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

Conformance for Numerical Limits: PASS

2.8.3 Run-Time Increasable Values

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

2.8.4 Run-Time Invariant Values (Possibly Indeterminate)

(IEEE Std 2003.1-1992 {4}) 04

UNUSED

(IEEE Std 2003.1-1992 {4}) 05

UNUSED

(IEEE Std 2003.1-1992 {4}) 06

UNUSED

(IEEE Std 2003.1-1992 {4}) 07

UNUSED

(IEEE Std 2003.1-1992 {4}) 08

UNUSED

Table 2-1 – Minimum Values

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

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

(IEEE Std 2003.1-1992 {4} 09

UNUSED

(IEEE Std 2003.1-1992 {4} D02

UNUSED

4 **TEST:** The values defined in Table 2-2 are equal to or less than those either defined in `<limits.h>` or provided by the *sysconf*() interface.

NOTE: This table is the same as Table 2-5 in IEEE Std 1003.1b-1993

Conformance for Numerical Limits: PASS

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

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

Name	Description	Minimum Value
{AIO_PRIO_DELTA_MAX}	The maximum amount by which a process can decrease its asynchronous I/O priority level from its own scheduling priority.	0
{ARG_MAX}	Maximum length of arguments for the *exec* functions, in bytes, including environment data.	{_POSIX_ARG_MAX}
{CHILD_MAX}	Maximum number of simultaneous processes per real user ID.	{_POSIX_CHILD_MAX}
{DELAYTIMER_MAX}	Maximum number of timer expiration overruns.	{_POSIX_DELAYTIMER_MAX}
{MQ_OPEN_MAX}	The maximum number of open message queue descriptors a process may hold.	{_POSIX_MQ_OPEN_MAX}
{MQ_PRIO_MAX}	The maximum number of message priorities supported by the implementation.	{POSIX_MQ_PRIO_MAX}
{OPEN_MAX}	Maximum number of files that one process can have open at any given time.	{_POSIX_OPEN_MAX}
{PAGESIZE}	Granularity in bytes of memory mapping and process memory locking.	1
{RTSIG_MAX}	Maximum number of realtime signals reserved for application use in this implementation.	{_POSIX_RTSIG_MAX}
{SEM_NSEMS_MAX}	Maximum number of semaphores that a process may have.	{_POSIX_SEM_NSEMS_MAX}
{SEM_VALUE_MAX}	The maximum value a semaphore may have.	{_POSIX_SEM_VALUE_MAX}
{SIGQUEUE_MAX}	Maximum number of queued signals that a process may send and have pending at the receiver(s) at any time.	{POSIX_SIGQUEUE_MAX}
{STREAM_MAX}	The number of streams that one process can have open at one time. If defined, it shall have the same value as {FOPEN_MAX} from the C Standard {2}.	{_POSIX_STREAM_MAX}
{TIMER_MAX}	Maximum number of timers per process supported by the implementation.	{_POSIX_TIMER_MAX}
{TZNAME_MAX}	The maximum number of bytes supported for the name of a time zone (not of the **TZ** variable).	{_POSIX_TZNAME_MAX}

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

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

2.8.5 Pathname Variable Values

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

2.8.6 Invariant Values

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

2.8.7 Maximum Values

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

Table 2-3 – Maximum Values

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

2.9 Symbolic Constants

(IEEE Std 2003.1-1992 {4} D01
UNUSED

(IEEE Std 2003.1-1992 {4} D02
UNUSED

(IEEE Std 2003.1-1992 {4} D03
UNUSED

(IEEE Std 2003.1-1992 {4} D04
UNUSED

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

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

2.9.1 Symbolic Constants for the *access*() Function

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

2.9.2 Symbolic Constant for the *lseek*() Function

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

2.9.3 Compile-Time Symbolic Constants for Portability Specifications

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

(IEEE Std 2003.1-1992 {4} R03
UNUSED

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

Table 2-4 – Compile-Time Symbolic Constants

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

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

(IEEE Std 2003.1-1992 {4} 05
UNUSED

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

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

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

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

2.9.4 Execution-Time Symbolic Constants for Portability Specifications

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Table 2-5 – Execution-Time Symbolic Constants

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

Section 3: Process Primitives

3.1 Process Creation and Execution

3.1.1 Process Creation

Function: *fork*()

3.1.1.1 Symbols

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

3.1.1.2 Description

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.1.1.3 Returns

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

3.1.1.4 Errors

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

3.1.2 Execute a File

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

3.1.2.1 Synopsis

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

3.1.2.1 Description

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

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

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

SETUP: Create locked pages in the address space of the process that will call *function*() and also map and lock the same pages into the address space of another process.
TEST: The memory page locks for memory pages that are mapped into the address space of other processes and locked by them are unaffected by a process that has locks on those same pages and that calls *function* ().
TR: Establish the memory locks using as many of the interfaces *mlockall*() and *mlock*() as are implemented.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 FOR: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF *PCTS_mmap* or *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect on memory locks in the *function*() fails does so in §3.1.2.2.
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_OPTION

4 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF *PCTS_mmap* or *PCTS_shm_open* **THEN**
TEST: Memory mappings created in a process are unmapped before the address space is rebuilt for the new process image after a call to *function*().
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_TEST, NO_OPTION

5 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF *PCTS_sched_setscheduler* and *PCTS_getscheduler* **THEN**
TEST: The policy and priority settings for the SCHED_FIFO and SCHED_RR scheduling policies are not changed for a process that calls *function*().
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_OPTION

D_2 FOR: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF *PCTS_sched_setscheduler* and *PCTS_getscheduler* **THEN**
TEST: The PCD.1b documents for scheduling policies other than SCHED_FIFO and SCHED_RR, the policy and priority settings after a call to *function*() in §3.1.2.2.
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_OPTION

6 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF *PCTS_timer_create and PCTS_timer_settime* **THEN**
TEST: Per-process timers created by the calling process are deleted before replacing the current process image with the new process image after a call to *function*().
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for execl, execv, execle, execve, execlp, execvp: PASS, NO_TEST, NO_OPTION

7 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()
IF *PCTS_timer_create and PCTS_timer_settime* **THEN**
TEST: Per-process timers created by the calling process are deleted after a call to *function*().
NOTE: This can be tested by setting a timer before the *function*() call and waiting in the new process image for the timer's signal. If no signal arrives after a sufficiently long time, the timer was destroyed or the equivalent. To determine of the timer was actually destroyed, try to create {TIMER_MAX} timers. If this is possible then the timer was destroyed.
ELSE *NO_OPTION*

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

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

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

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

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

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

D_4 FOR: *execl*(), *execv*(), *execle*(), *execve*(), *execlp*(), *execvp*()

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

3.1.2.3 Returns

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

3.1.2.4 Errors

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

3.2 Process Termination

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

3.2.1 Wait for Process Termination

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

3.2.1.1 Synopsis

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

3.2.1.2 Description

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

3.2.1.3 Returns

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

3.2.1.4 Errors

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

3.2.2 Terminate a Process

Function: *_exit*()

3.2.2.1 Synopsis

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

3.2.2.2 Description

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

Conformance for _exit: PASS, NO_TEST_SUPPORT, NO_OPTION

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

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

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

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

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

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

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

ELSE *NO_OPTION*

Conformance for _exit: PASS, NO_OPTION

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

Conformance for _exit: PASS

3.2.2.3 Returns

There are no requirements for conforming implementations in this subclause.

3.3 Signals

3.3.1 Signal Concepts

3.3.1.1 Signal Names

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 the reader's following the correspondence between assertions in this standard and the requirements specified in POSIX.1b {3}.

Table 3-1 – Required Signals

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

NOTE: The default actions are

1 Abnormal termination of the process.

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

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

Conformance for signal.h: PASS

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

Table 3-2 – Job Control Signals

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

NOTE: The default actions are

2 Ignore the signal.

3 Stop the process.

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

Table 3-3 – Memory Protection Signals

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

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

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

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

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

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

3.3.1.2 Signal Generation and Delivery

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

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

Conformance for signal.h: PASS

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

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

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

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

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

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

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

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

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

Conformance for signal.h: PASS

M_GA_sigPending(function)=
SETUP: Cause the *function*() interface to generate a signal after having set an application-specified value to be transmitted along with the signal.
TEST: After a signal is generated by the *function*() interface the signal shall be marked pending, and, if the SA_SIGINFO flag is set for that signal, the signal shall be queued to the process along with the application-specified signal value.
TR: Test for all catchable signals both with SA_SIGINFO set and not set.

GA_sigPending
FOR: *sigqueue*(), *lio_listio*(), *timer_create*(), *mq_notify*(), *aio_read*(), *aio_write*(), and *aio_fsync*()
M_GA_sigPending(function)
Conformance for signal.h: PASS, NO_OPTION

M_GA_sigPendingQueued(function)=
SETUP: Cause the *function*() interface to generate a series of signals after having set a different application-specified value for each signal to be transmitted along with the signal.
TEST: Multiple occurrences of signals is generated by the *function*() interface with the SA_SIGINFO flag set for that signal will be queued in FIFO order.
TR: Test for all catchable signals.

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

D_4 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether signals generated by the *sigqueue*() function or any signal-generating function that supports the specification of an application-defined value are queued when the SA_SIGINFO flag is not set for that signal does so in §3.3.1.2.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

M_GA_queuedAndRegularSignals(function)=
SETUP: Cause the *function*() interface to generate a signal after having set a different application-specified value for each signal to be transmitted along with the signal.
TEST: Signals generated by the *kill*() function or other events that cause signals to occur, such *alarm*() timer expiration or terminal activity, and for which the implementation does not support queuing, have no effect on signals already queued for the same signal number.
TR: Test for all catchable signals and all of the conditions specified in the **TEST.**

GA_queuedAndRegularSignals
FOR: *sigqueue*(), *lio_listio*(), *timer_create*(), *mq_notify*(), *aio_read*(), *aio_write*(), and *aio_fsync*()
M_GA_queuedAndRegularSignals(function)
Conformance for signal.h: PASS, NO_OPTION

9 **IF** *_POSIX_REALTIME_SIGNALS* **THEN**

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

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

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

3.3.1.3 Signal Actions

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

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

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

D_5 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of a process after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that was not generated by the *kill*() function, the *sigqueue*() function, or the *raise*() function as defined by the C Standard {2} does so in §3.3.1.3.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

15 **IF** {_POSIX_REALTIME_SIGNALS} **THEN**
IF *PCTS_sigqueue* **THEN**
TEST: Setting a signal action to SIG_IGN for a signal that is pending causes the pending signal to be discarded, whether or not it is blocked; in addition, any queued values pending are discarded.
TR: Test for all catchable signals consistent with the implemented options.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **IF** {_POSIX_REALTIME_SIGNALS} **THEN**
IF *PCTS_sigqueue* **THEN**
TEST: If the implementation limits the number of outstanding queued signals to {SIGQUEUE_MAX}, this test can verify that signal can be sent after setting the action to SIG_IGN in the target process. Setting a signal action to SIG_IGN for a signal that is pending causes the resources used to queue the pending signal and its associated value to be released and made available to queue other signals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

17 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**
IF *PCTS_sigqueue* **THEN**
TEST: On delivery of a signal specified in Table 3-3, the receiving process executes the signal-catching function at the specified address.
TR: Test for all signals in Table 3-3.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_sigqueue* **THEN**
SETUP: Clear the SA_SIGINFO flag and establish a signal-catching function based on the following prototype:

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

TEST: The signal-catching function is passed *signo* as a parameter.
TR: Test for all catchable signals consistent with the implemented options.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

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

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

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

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

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

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

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

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

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

Conformance for signal.h: PASS

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

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

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

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

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

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

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

Conformance for signal.h: PASS

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

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

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

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

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

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

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

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

24 **IF** *PCTS_timer_settime* **THEN**

IF *PCTS_timer_create* **THEN**

SETUP: With the SA_SIGINFO flag set, declare a signal-catching function and choose an application-specific value to be passed to it. Then generate a signal for that signal-catching function to handle by setting up a timer expiration for a timer set by *timer_settime*().

TEST: The *se_code* member of the *siginfo_t* structure contains SI_TIMER, a code identifying the cause of the signal generation as the expiration of a timer set by *timer_settime*(), and the *si_signo* member of the *siginfo_t* structure contains the signal number and it is the same as the *signo* parameter passed to the signal-catching function.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

25 **IF** {_POSIX_ASYNCHRONOUS_IO} and {POSIX_REALTIME_SIGNALS) **THEN**

SETUP: With the SA_SIGINFO flag set, declare a signal-catching function and choose an application-specific value to be passed to it. Then generate a signal for that signal-catching function to handle by completing an asynchronous I/O request.

TEST: The *si_code* member of the *siginfo_t* structure contains SI_ASYNCIO, a code identifying the cause of the signal generation as the completion of an asynchronous I/O request**,** and the *si_signo* member of the *siginfo_t* structure contains the signal number and it is the same as the *signo* parameter passed to the signal-catching function.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

26 **IF** {_POSIX_MESSAGE_PASSING} and {_POSIX_REALTIME_SIGNALS} **THEN**

SETUP: With the SA_SIGINFO flag set, declare a signal-catching function and choose an application-specific value to be passed to it. Then generate a signal for that signal-catching function to handle by making a message arrive on an empty message queue.

TEST: The *si_code* member of the *siginfo_t* structure contains the code SI_MESGQ, which means the signal was generated by the arrival of a message on an empty message queue, and the *si_signo* member of the *siginfo_t* structure contains the signal number and it is the same as the *signo* parameter passed to the signal-catching function.

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

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

1. The *kill*() function.
2. The *raise*() or *abort*() functions as defined in the C Standard {2}, if they set *si_code* to SI_USER.
3. The *sigqueue*() function.
4. The *timer_settime*() function.
5. The completion of an asynchronous I/O request.
6. The arrival of a message on an empty message queue in §3.3.1.3.

Conformance for signal h: PASS

27 **IF** *PCTS_MORE_SA_SIGINFO_SIGNALS* and {_POSIX_REALTIME_SIGNALS) **THEN**
SETUP: Establish a signal-catching function with the SA_SIGINFO flag set. Then generate that signal by a means other than calling *kill*(), *raise*() and *abort*() (if they set to *sigqueue*(), or *timer_settime*() or by completion of an asynchronous I/O request or by the arrival of a message on an empty message queue.
TEST: The *si_code* is set to an implementation-defined value that is not equal to any of the values defined in IEEE Std 1003.1b-1993 for *si_code*.
TR: Test for each such implementation-defined value.
NOTE: It is possible to perform this test by calling a target system-specific function that contains the tests.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

28 **IF** {_POSIX_REALTIME_SIGNALS} **THEN**
IF *PCTS_sigqueue* **THEN**
SETUP: Create a signal so that *si_code* contains the value SI_QUEUE.
TEST: The *si_value* contains the application-specified signal value.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

29 **IF** {_POSIX_REALTIME_SIGNALS} **THEN**
IF *PCTS_timer_settime* and *PCTS_timer_create* **THEN**
SETUP: Create a signal so that *si_code* contains the value SI_TIMER.
TEST: The *si_value* contains the application-specified signal value.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_TEST_SUPPORT, NO_OPTION

30 **IF** {_POSIX_ASYNCHRONOUS_IO} **THEN**
SETUP: Create a signal so that *si_code* contains the value SI_ASYNCIO.
TEST: The *si_value* contains the application-specified signal value.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

31 **IF** {_POSIX_MESSAGE_PASSING) **THEN**
SETUP: Create a signal so that *si_code* contains the value SI_MESGQ.
TEST: The *si_value* contains the application-specified signal value.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

D_8 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the *context* parameter to a signal-catching function does so in §3.3.1.3.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

D_9 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of a process after it returns normally from a signal-catching function for a SIGFPE, SIGHILL, SIGSEGV, or SIGBUS signal that was not generated by the *kill* () function, the *sigqueue*() function, or the *raise*() function as defined by the C Standard {2} does so in §3.3.1.3.
ELSE *NO_OPTION*
Conformance for signal.h: PASS, NO_OPTION

32 **IF** *PCTS_aio_error* **THEN**

TEST: The function *aio_error*() is reentrant with respect to signals (that is, applications may invoke them, without restriction, from signal-catching functions).

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

33 **IF** *PCTS_aio_return* **THEN**

TEST: The function *aio_return*() is reentrant with respect to signals (that is, applications may invoke them, without restriction, from signal-catching functions).

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

34 **IF** *PCTS_aio_suspend* **THEN**

TEST: The function *aio_suspend*() is reentrant with respect to signals (that is, applications may invoke them, without restriction, from signal-catching functions).

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

35 **IF** *PCTS_clock_gettime* **THEN**

TEST: The function *clock_gettime*() is reentrant with respect to signals (that is, applications may invoke them, without restriction, from signal-catching functions).

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

36 **IF** *PCTS_fdatasync* **THEN**

TEST: The function *fdatasync*() is reentrant with respect to signals (that is, applications may invoke them, without restriction, from signal-catching functions).

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

37 **IF** *PCTS_sem_post* **THEN**

TEST: The function *sem_post*() is reentrant with respect to signals (that is, applications may invoke them, without restriction, from signal-catching functions).

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

38 **IF** *PCTS_sig_queue* **THEN**

TEST: The function *sig_queue*() is reentrant with respect to signals (that is, applications may invoke them, without restriction, from signal-catching functions).

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

39 **IF** *PCTS_timer_getoverrun* **THEN**

TEST: The function *timer_getoverrun*() is reentrant with respect to signals (that is, applications may invoke them, without restriction, from signal-catching functions).

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

40 **IF** *PCTS_timer_gettime* **THEN**

TEST: The function *timer_gettime*() is reentrant with respect to signals (that is, applications may invoke them, without restriction, from signal-catching functions).

ELSE *NO_OPTION*

Conformance for signal.h: PASS, NO_OPTION

41 **IF** *PCTS_timer_settime* **THEN**

TEST: The function *timer_settime*() is reentrant with respect to signals (that is, applications may invoke them, without restriction, from signal-catching functions).

ELSE *NO_OPTION*

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**

1 **SETUP:** Call *sigaction*() with the SA_SIGINFO flag cleared in the *sa_flags* field of the *sigaction* structure to establish a signal-catching function.
TEST: The *sa_handler* field identifies the action to be associated with the specified signal.
TR: Test for all catchable signals consistent with implemented options.
Conformance for sigaction: PASS

2 **IF** {_POSIX_REALTIME_SIGNALS} **THEN**
SETUP: Call *sigaction*() with the SA_SIGINFO flag set in the *sa_flags* field of the *sigaction* structure to establish a signal-catching function.
TEST: The *sa_sigaction* field specifies a signal-catching function.
TR: Test for all catchable signals consistent with implemented options.
ELSE *NO_OPTION*
Conformance for sigaction: PASS, NO_OPTION

3 **SETUP:** Call *sigaction*() with the SA_SIGINFO bit cleared and the *sa_handler* field specifying a signal-catching function and perform the test. Then call it again with the SA_SIGINFO bit set and perform the test.
TEST: The *sa_mask* field identifies a set of signals that are added to the signal mask of the process before the signal-catching function is invoked.
TR: Test for all catchable signals consistent with implemented options. Do not test for SIGKILL and SIGSTOP.
Conformance for sigaction: PASS

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

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

Conformance for sigaction: PASS

D_1 TEST: The PCD.1b documents the disposition of subsequent occurrences of *sig* when it is already pending and the signal-catching function was established with SA_SIGINFO not set in *sa_flags* in §3.3.4.2.
Conformance for sigaction: PASS

5 **IF** {POSIX_REALTIME_SIGNALS} **THEN**
SETUP: Set SA_SIGINFO in *sa_flags* and establish a signal-catching function by calling *sigaction*().
TEST: Subsequent occurrences of *sig* generated by *sigqueue*() or as a result of any signal-generating function that supports the specification of an application-defined value – when *sig* is already pending – is queued in FIFO order until delivered; the signal-catching function is invoked with three arguments; and the application specified value is passed to the signal-catching function as the *si_value* member of the *siginfo_t* structure.
TR: Test for the following functions:

1. If {_POSIX_REALTIME_SIGNALS} is defined then *kill*().
2. If *PCTS_sigqueue* then *sigqueue*().
3. If *PCTS_lio_listio* then *lio_listio*().
4. If *PCTS_timer_create* and *PCTS_timer_settime* then *timer_settime*().

5. If *PCTS_mq_open* and *PCTS_mq_send* and *PCTS_mq_notify* then *mq_notify*().

ELSE *NO_OPTION*
Conformance for sigaction: PASS, NO_OPTION

3.3.4.3 Returns

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

3.3.4.4 Errors

6 **IF** {POSIX_REALTIME_SIGNALS} is not supported **THEN**
TEST: A call to *sigaction*() with the SA_SIGINFO bit flag set in the *sa_flags* field of the *sigaction* structure returns a value of -1 and sets *errno* to [ENOTSUP].
ELSE *NO_OPTION*
Conformance for sigaction: PASS, NO_OPTION

3.3.5 Examine and Change Blocked Signals

Function: *sigprocmask*()

3.3.5.1 Synopsis

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

3.3.5.2 Description

D_1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of generating a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal while they are blocked, when the signal was not generated by the *kill*() function, the *sigqueue*() function, or the *raise*() function as defined by the C Standard {2} does so in §3.3.5.2.
ELSE *NO_OPTION*
Conformance for sigprcmask: PASS, NO_OPTION

3.3.5.3 Returns

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

3.3.5.4 Errors

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

3.3.6 Examine Pending Signals

Function: *sigpending*()

3.3.6.1 Synopsis

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

3.3.6.2 Description

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

3.3.6.3 Returns

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

3.3.6.4 Errors

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

3.3.7 Wait for a Signal

Function: *sigsuspend*()

3.3.7.1 Synopsis

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

3.3.7.2 Description

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

3.3.7.3 Returns

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

3.3.7.4 Errors

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

3.3.8 Synchronously Accept a Signal

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

3.3.8.1 Synopsis

1

M_GA_stdC_proto_decl(int; sigwaitinfo: const sigset_t *set, siginfo_t *info; signal.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sigwaitinfo: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sigwaitinfo; signal.h;;;)
SEE: Assertion GA_commonc_int_result_decl in §2.7.3
Conformance for sigwaitinfo: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sigwaitinfo; signal.h;;;)
SEE: Assertion GA_macro_result in §1.3.4
Conformance for sigwaitinfo: PASS, NO_OPTION

4

M_GA_macro_args(sigwaitinfo; signal.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sigwaitinfo: PASS, NO_OPTION

5

M_GA_stdC_proto_decl(int; sigtimedwait; , const sigset_t *set, siginfo_t *info, const struct timespec *timeout; signal.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sigtimedwait: PASS[5, 6], NO_OPTION

6

M_GA_commonC_int_result_decl(sigtimedwait; , const sigset_t *set, siginfo_t *info, const struct timespec *timeout; signal.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sigtimedwait: PASS, NO_OPTION

7

M_GA_macro_result_decl(int; sigtimedwait; signal.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sigtimedwait: PASS, NO_OPTION

8

M_GA_macro_args (sigtimedwait; signal.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sigtimedwait: PASS, NO_OPTION

3.3.8.2 Description

9 **IF** *PCTS_sigwaitinfo* **THEN**
TEST: The function *sigwaitinfo*() waits for the pending signal from the set of signals specified by the *set* parameter and returns the selected signal number.
TR : Test for only one signal at a time by specifying them in *set* then send two signals, one f which is the signal specified in *set*. Test this for signals in the range SIGRTMIN to SIGRTMAX. Then do the same test separately for all non-realtime signals supported by the implementation.
ELSE *NO_OPTION*
Conformance for sigwaitinfo: PASS, NO_OPTION

10 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**
TEST: When there are multiple pending signals in the range SIGRTMIN to SIGRTMAX which have been specified in the *set* argument to *function*() the signal number returned is the lowest numbered one.
NOTE: This can be tested by creating two processes, one to send signals and one to wait for them by calling *function*(). Block the multiple signals to be tested. Then send those signals. Then wait for signals by calling *function*().
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

D_1 FOR: *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* and a PCD.1 documents the following **THEN**
TEST: A PCD.1b that documents the selection order between realtime and nonrealtime signals, or between multiple pending nonrealtime signals does so in §3.3.8.2.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

11 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**
SETUP: Call *function*() with no signal in *set* pending at the time of the call.
TEST: The process calling *function*() is suspended until one or more signals in *set* become pending, in which case the selected signal number is returned, or until it is interrupted by an unblocked, caught signal.
TR: Test for both cases: when no signals are pending and when the call is interrupted by an unblocked, caught signal.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

12 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**

SETUP: Call *function*() with the *info* argument non-**NULL**
TEST: The selected signal number is stored in the *si_signo* member, and the cause of the signal is stored in the *si_code* member of the *siginfo_t* structure pointed to by the *info* argument and the selected signal number is returned.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

13 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**
SETUP: Queue different application-specified values for the signal to be selected by a call to *function*() with a non-**NULL** value in the *info* argument.
TEST: The first queued value is dequeued and stored in the *si_value* member of the *siginfo_t* structure pointed to by the *info* argument after a call to *function*() with a non-**NULL** *info* argument and the selected signal number is returned.
TR: Test that all queued values are retrieved in queued order.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

14 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**
TEST: The system resource used to queue the signal selected by *function*() is released and made available to queue other signals when *sigwaitinfo*() returns with the selected signal number.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

D_2 FOR: *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the content of the *si_value* member of the *info* argument when no value is queued does so in §3.3.8.2.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

15 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF *PCTS_function* **THEN**
TEST: After dequeuing all pending signals selected by *function*(), the pending indication for each signal is reset.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

16 **IF** *PCTS_sigtimedwait* **THEN**
SETUP: Call *sigtimedwait*() when none of the signals specified by the *set* arguments are pending.
TEST: The function *sigtimedwait*() waits for the time interval specified in the *timespec* structure referenced by *timeout* before returning.
ELSE *NO_OPTION*
Conformance for sigtimedwait: PASS, NO_OPTION

17 **IF** *PCTS_sigtimedwait* **THEN**
SETUP: Call *sigtimedwait*() with the *timespec* structure pointed to by *timeout* being zero-valued and none of the signals specified by *set* pending.
TEST: The *sigtimedwait*() returns immediately with an error.
ELSE *NO_OPTION*
Conformance for sigtimedwait: PASS, NO_OPTION

D_3 IF *PCTS_sigtimedwait* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior with the *timeout* argument to *sigtimedwait*() is the **NULL** pointer does so in §3.3.8.2.
ELSE *NO_OPTION*

Conformance for sigtimedwait: PASS, NO_OPTION

18 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF ***PCTS_function*** **THEN**
TEST: While *function*() is waiting and a signal occurs that is eligible for delivery (i.e., not blocked by the process signal mask), that signal is handled asynchronously and the wait is interrupted.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

D_4 FOR: *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF not {_POSIX_REALTIME_SIGNALS} and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents its support or lack of support for *function*() does so in §3.3.8.2.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

3.3.8.3 Returns

R_1 TEST: Upon successful completion (that is, one of the signals specified by *set* is pending or is generated) *sigwaitinfo*() and *sigtimedwait*() return the selected signal number.
SEE: All assertions in §3.3.8.2.

R_2 TEST: The functions *sigwaitinfo*() and *sigtimedwait*() return a value of -1 and set *errno* to indicate the error.
SEE: All assertions in §3.3.8.4.

3.3.8.4 Errors

19 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF ***PCTS_function*** **THEN**
SETUP: Call *function*() to wait on an unblocked signal that will be caught.
TEST: The function *function*() returns -1 and sets *errno* to [EINTR] when the wait was interrupted by an unblocked, caught signal.
TR: Test for each catchable signal consistent with the implemented options.
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

20 **FOR:** *PCTS_sigwaitinfo* and *PCTS_sigtimedwait*
IF not *PCTS_function* **THEN**
TEST: A call to *function*() returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sigwaitinfo, sigtimedwait: PASS, NO_OPTION

21 **IF** not *PCTS_sigtimedwait* **THEN**
TEST: A call to *sigtimedwait*() returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sigtimedwait: PASS, NO_OPTION

22 **IF** *PCTS_sigwaitinfo* and *PCTS_SIGTIMEDWAIT_VALUE* **THEN**
TEST: The *sigtimedwait*() function when called with a timeout argument specifying a *tv_nsec* value less than zero or greater than or equal to 1000 million, and when no signal is pending in *set* and it is necessary to wait, returns -1 and sets *errno* to [EINVAL].
ELSE *NO_OPTION*
Conformance for sigwaitinfo: PASS, NO_OPTION

D_5 IF *PCTS_sigwaitinfo* and *PCTS_SIGTIMEDWAIT_VALUE* **THEN**

TEST: A PCD.1b that documents the detection of a *timeout* argument specifying a *tv_nsec* value less than zero or greater than or equal to 1000 million and the conditions under which the error is detected does so in §3.3.8.4.

ELSE *NO_OPTION*

Conformance for sigwaitinfo: PASS, NO_OPTION

3.3.9 Queue a Signal to a Process

Function: *sigqueue*()

3.3.9.1 Synopsis

1

M_GA_stdC_proto_decl(int; sigqueue; pid_t pid, int signo, const union sigval value; signal.h;;;)

SEE: Assertion GA_stC_proto_decl in §2.7.3

Conformance for sigqueue: PASS[1,2], NO_OPTION

2

MG_GA_commonC_int_result_decl(sigqueue; signal.h;;;)

SEE: Assertion GA_commonC_int_result_decl in §2.7.3

Conformance for sigqueue: PASS[1,2], NO_OPTION

3

MG_GA_macro_result_decl(int; sigqueue; signal.h;;;)

SEE: Assertion GA_macro_result_decl in §1.3.4

Conformance for sigqueue: PASS, NO_OPTION

4 ***MG_GA_macro_args(sigqueue; signal.h;;;)***

SEE: Assertion GA_macro_args in §2.7.3

Conformance for sigqueue: PASS, NO_OPTION

3.3.9.2 Description

sigqueue_priv()=

Create two processes, one to send the signal and one to receive it, such that if *PCTS_GAP_sigqueue*, the real or effective user ID of the calling process does not match the real or effective user ID of the process to which the signal is being sent; or if {_POSIX_SAVED_IDS} is the real or effective user ID of the calling process matches the real or saved set-user-ID of the process to which the signal is being sent; otherwise, the real or effective user ID of the calling process matches the real or effective user ID of the process to which the signal is being sent.

5 **IF** *PCTS_sigqueue* **THEN**

SETUP:

***sigqueue_priv*()**

TEST: The *sigqueue*() function causes the signal specified by *signo* to be sent with the value specified by *value* to the process specified by *pid*.

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

ELSE *NO_OPTION*

Conformance for sigqueue: PASS, NO_OPTION

6 **IF** *PCTS_sigqueue* **THEN**
IF {SIGQUEUE_MAX} <= *PCTS_SIGQUEUE_MAX* **THEN**
SETUP:
sigqueue_priv()
TEST: The *sigqueue*() function can queue a total of {SIGQUEUE_MAX} signals to one or more processes.
TR: Test for one and two processes.

Test for all catchable signals consistent with implemented options. Also, test for the supported combinations of real and effective user IDs as well as for appropriate privilege and saved set-user-ID, if they are supported by the implementation.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_sigqueue* **THEN**
IF {SIGQUEUE_MAX} > *PCTS_SIGQUEUE_MAX* **THEN**
SETUP:
sigqueue_priv()
TEST: The *sigqueue*() function can queue a total of *PCTS_SIGQUEUE_MAX* signals to one or more processes.
TR: Test for one and two processes.

Test for all catchable signals consistent with implemented options. Also, test for the supported combinations of real and effective user IDs as well as for appropriate privilege and saved set-user-ID, if they are supported by the implementation.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_sigqueue* **THEN**
SETUP:
sigqueue_priv()
TEST: If the *signo* argument is *sigqueue*() is zero (the null signal), error checking is performed but no signal is actually sent.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_OPTION

9 **IF** *PCTS_sigqueue* **THEN**
SETUP:
sigqueue_priv()
Block the signal specified in *signo* for the receiving process and clear the SA_SIGINFO flag for it.
TEST: A call to *sigqueue*() returns immediately.
TR: Test for all catchable signals consistent with implemented options. Also, test for the supported combinations of real and effective user IDs as well as for appropriate privilege and saved set-user-ID, if they are supported by the implementation.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_OPTION

10 **IF** *PCTS_sigqueue* **THEN**
SETUP:
sigqueue_priv()
Block the signal specified in *signo* for the receiving process and set the SA_SIGINFO flag for it.
TEST: A call to *sigqueue*() returns immediately, and if the resources are available to queue the signal, the signal is left queued and pending.

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

ELSE *NO_OPTION*

Conformance for sigqueue: PASS, NO_OPTION

11 **IF** *PCTS_sigqueue* **THEN**

SETUP:

sigqueue_priv()

Do not set SA_SIGINFO for *signo*

TEST: The *signo* is sent at least once to the receiving process.

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

ELSE *NO_OPTION*

Conformance for sigqueue: PASS, NO_OPTION

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

TEST: A PCD.1b that documents whether or not the *value* parameter is sent to the receiving process as a result of calling the *sigqueue*() function when SA_SIGINFO is not set for *signo* and the conditions under which the *value* is sent does so in §3.3.9.2.

ELSE *NO_OPTION*

Conformance for sigqueue: PASS, NO_OPTION

12 **IF** *PCTS_sigqueue* **THEN**

SETUP:

Set *pid* so that it causes *signo* to be generated for the sending process, and do not block signo.

TEST: Either *signo* or at least the pending, unblocked signal with the lowest number shall be delivered to the sending process before the *sigqueue*() function returns.

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

ELSE *NO_OPTION*

Conformance for sigqueue: PASS, NO_OPTION

D-2 **IF** not {_POSIX_REALTIME_SIGNALS} and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents its support or lack of support for *sigqueue*() does so in §3.3.8.2.

ELSE *NO_OPTION*

Conformance for sigqueue: PASS, NO_OPTION

3.3.9.3 Returns

R-1 **IF** *PCTS_sigqueue* **THEN**

TEST: Upon successful completion of a call to *sigqueue*(), the specified signal has been queued, and the function returns a value of zero.

ELSE *NO_OPTION*

SEE: Assertions in §3.3.9.2.

R-2 **IF** *PCTS_sigqueue* **THEN**

TEST: An unsuccessful call to *sigqueue*() returns a value of -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: Assertions in §3.3.9.4.

3.3.9.4 Errors

13 **IF** *PCTS_sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [EAGAIN] when no resources are available to queue the signal.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST, NO_OPTION

14 **IF** *PCTS_sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [EAGAIN] when the process has already queued {SIGQUEUE_MAX} signals that are still pending at the receiver(s).
NOTE: There is no known portable test method for this assertion.
TR: Test limit for signals sent to one process and to two processes.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_OPTION

15 **IF** *PCTS_sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [EAGAIN] when a systemwide resource limit has been exceeded.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST, NO_OPTION

16 **IF** *PCTS_sigqueue* **THEN**
IF *PCTS_INVALID_SIGNAL* or *PCTS_UNSUPPORTED_SIGNAL* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [EINVAL] when the value of the *signo* argument is an invalid or unsupported signal number.
TR: Test for both invalid and unsupported signals, if each exists.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST_SUPPORT, NO_OPTION

17 **IF** not *PCTS_sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [ENOSYS] when the function *sigqueue*() is not supported by this implementation.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_OPTION

18 **IF** *PCTS_sigqueue* **THEN**
IF ***PCTS_RAP_****sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [EPERM] when the process does not have the appropriate privilege to send the signal to the receiving process.
TR: Use *sigqueue_priv*().
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_sigqueue* **THEN**
TEST: A call to *sigqueue*() returns -1 and sets *errno* to [ESDRCH] when the process *pid* does not exist.
ELSE *NO_OPTION*
Conformance for sigqueue: PASS, NO_OPTION

3.4 Timer Operations

There are no requirements for conforming implementations in this subclause.

3.4.1 Schedule Alarm

Function: *alarm*()

3.4.1.1 Synopsis

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

3.4.1.2 Description

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

3.4.1.3 Returns

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

3.4.1.4 Errors

There are no requirements for conforming implementations in this subclause.

3.4.2 Suspend Process Execution

Function: *pause*()

3.4.2.1 Synopsis

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

3.4.2.2 Description

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

3.4.2.3 Returns

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

3.4.2.4 Errors

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

3.4.3 Delay Process Execution

Function: *sleep*()

3.4.3.1 Synopsis

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

3.4.3.2 Description

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

3.4.3.3 Returns

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

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

4.8 Configurable System Variables

4.8.1 Get Configurable System Variables

Function: *sysconf*()

4.8.1.1 Synopsis

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

4.8.1.2 Description

NOTE: There is no Table 4-1 in this section. Table 4-2 in this standard is the same as Table 4-2 in POSIX.1b {3}.

(IEEE Std 2003.1-1992 {4} 04
UNUSED

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

2 **FOR:** {AIO_LISTIO_MAX}, {AIO_MAX}, {AIO_PRIO_DELTA_MAX}, {DELAYTIMER_MAX}, {MQ_OPEN_MAX}, {MQ_PRIO_MAX}, {PAGESIZE}, {RTSIG_MAX}, {SEM_NSEMS_MAX}, {SEM_VALUE_MAX}, {SIGQUEUE_MAX}, {TIMER_MAX},
IF CONSTANT is defined in `<limits.h>` **THEN**
SETUP: Include the header `<limits.h>`
TEST: A call *sysconf(_SC_CONSTANT)* either returns -1 without changing the value of *errno* or returns a value greater than or equal to {CONSTANT}.
NOTE: The assertion is tested once for each constant specified in the FOR clause. The assertion is to be read by substituting CONSTANT with the current constant specified in the FOR clause. The name of the compound constant also is to be substituted for each occurrence in the constructs *_SC_CONSTANT* and *_POSIX_CONSTANT*.
ELSE *NO_TEST_SUPPORT*
Conformance for sysconf: PASS, NO_TEST_SUPPORT

3 **FOR:** {AIO_LISTIO_MAX}, {AIO_MAX}, {AIO_PRIO_DELTA_MAX}, {DELAYTIMER_MAX}, {MQ_OPEN_MAX}, {MQ_PRIO_MAX}, {PAGESIZE}, {RTSIG_MAX}, {SEM_NSEMS_MAX}, {SEM_VALUE_MAX}, {SIGQUEUE_MAX}, {TIMER_MAX},

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

SETUP: Include the header `<limits.h>`

TEST: A call *sysconf(_SC_CONSTANT)* either returns -1 without changing the value of *errno* or returns a value greater than or equal to {_POSIX_CONSTANT}.

NOTE: The assertion is tested once for each constant specified in the FOR clause. The assertion is to be read by substituting CONSTANT with the current constant specified in the FOR clause. The name of the compound constant also is to be substituted for each occurrence in the constructs *_SC_CONSTANT* and *_POSIX_CONSTANT*.

ELSE *NO_TEST_SUPPORT*

Conformance for sysconf: PASS, NO_TEST_SUPPORT

4 **FOR:** {_POSIX_ASYNCHRONOUS_IO}, {_POSIX_FSYNC}, {_POSIX_MAPPED_FILES}, {_POSIX_MEMLOCK}, {_POSIX_MEMLOCK_RANGE}, {_POSIX_MEMORY_PROTECTION}, {_POSIX_MESSAGE_PASSING}, {_POSIX_PRIORITIZED_IO}, {_POSIX_PRIORITY_SCHEDULING}, {_POSIX_REALTIME_SIGNALS}, {_POSIX_SEMAPHORES}, {_POSIX_SHARED_MEMORY_OBJECTS}, {POSIX_SYNCHRONIZED_IO}, {_POSIX_TIMERS},

IF CONSTANT is defined in `<unistd.h>` **THEN**

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

TEST: A call *sysconf(_SC_CONSTANT)* returns a value other than -1.

NOTE: The assertion is tested once for each constant specified in the FOR clause. The assertion is to be read by substituting CONSTANT with the current constant specified in the FOR clause. The name of the compound constant also is to be substituted for each occurrence in the constructs *_SC_CONSTANT* and *_POSIX_CONSTANT*.

ELSE *NO_TEST_SUPPORT*

Conformance for sysconf: PASS, NO_TEST_SUPPORT

Table 4-2 – Configurable System Variables

Variable	*name* Value
{AIO_LISTIO_MAX}	{_SC_AIO_LISTIO_MAX}
{AIO_MAX}	{_SC_AIO_MAX}
{AIO_PRIO_DELTA_MAX}	{_SC_AIO_PRIO_DELTA_MAX}
{ART_MAX}	{_SC_ARG_MAX}
{CHILD_MAX}	{_SC_CHILD_MAX}
clock ticks/second	{_SC_CLK_TCK}
{DELAYTIMER_MAX}	{_SC_DELAYTIMER_MAX}
{MQ_OPEN_MAX}	{_SC_MQ_OPEN_MAX}
{MQ_PRIO_MAX}	{_SC_MQ_PRIO_MAX}
{NGROUPS_MAX}	{_SC_NGROUPS_MAX}
{OPEN_MAX}	{_SC_OPEN_MAX}
{PAGESIZE}	{_SC_PAGESIZE}
{RTSIG_MAX}	{_SC_RTSIG_MAX}
{SEM_NSEMS_MAX}	{_SC_SEM_NSEMS_MAX}
{SEM_VALUE_MAX}	{_SC_SEM_VALUE_MAX}
{SIGQUEUE_MAX}	{_SC_SIGQUEUE_MAX}
{STREAM_MAX}	{_SC_STREAM_MAX}
{TIMER_MAX}	{_SC_TIMER_MAX}
{TZNAME__MAX}	{_SC_TZNAME_MAX}
{_POSIX_ASYNCHRONOUS_IO}	{_SC_ASYNCHRONOUS_IO}
{_POSIX_FSYNC}	{_SC_FSYNC}
{_POSIX_JOB_CONTROL}	{_SC_JOB_CONTROL}
{_POSIX_MAPPED_FILES}	{_SC_MAPPED_FILES}
{_POSIX_MEMLOCK}	{_SC_MEMLOCK}
{_POSIX_MEMLOCK_RANGE}	{_SC_MEMLOCK_RANGE}
{_POSIX_MEMORY_PROTECTION}	{_SC_MEMORY_PROTECTION}
{_POSIX_MESSAGE_PASSING}	{_SC_MESSAGE_PASSING}
{_POSIX_PRIORITIZED_IO}	{_SC_PRIORITIZED_IO}
{_POSIX_PRIORITY_SCHEDULING}	{_SC_PRIORITY_SCHEDULING}
{_POSIX_REALTIME_SIGNALS}	{_SC_REALTIME_SIGNALS}
{_POSIX_SAVED_IDS}	{_SC_SAVED_IDS}
{_POSIX_SEMAPHORES}	{_SC_SEMAPHORES}
{_POSIX_SHARED_MEMORY_OBJECTS}	{_SC_SHARED_MEMORY_OBJECTS}
{_POSIX_SYNCHRONIZED_IO}	{_SC_SYNCHRONIZED_IO}
{_POSIX_TIMERS}	{_SC_TIMERS}
{_POSIX_VERSION}	{_SC_VERSION}

5 **FOR:** {_POSIX_ASYNCHRONOUS_IO}, {_POSIX_FSYNC}, {_POSIX_MAPPED_FILES}, {_POSIX_MEMLOCK}, {_POSIX_MEMLOCK_RANGE}, {_POSIX_MEMORY_PROTECTION}, {_POSIX_MESSAGE_PASSING}, {_POSIX_PRIORITIZED_IO}, {_POSIX_PRIORITY_SCHEDULING}, {_POSIX_REALTIME_SIGNALS}, {_POSIX_SEMAPHORES}, {_POSIX_SHARED_MEMORY_OBJECTS}, {POSIX_SYNCHRONIZED_IO}, {_POSIX_TIMERS},

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

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

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

NOTE: The assertion is tested once for each constant specified in the FOR clause. The assertion is to be read by substituting CONSTANT with the current constant specified in the FOR clause. The name of the compound constant also is to be substituted for each occurrence in the constructs *_SC_CONSTANT* and *_POSIX_CONSTANT*.

ELSE *NO_TEST_SUPPORT*
Conformance for sysconf: PASS, NO_TEST_SUPPORT

4.8.1.3 Returns

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

4.8.1.4 Errors

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

4.8.1.5 Special Symbol {CLK_TCK}

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

Section 5: Files and Directories

There are no requirements for conforming implementations in this subclause.

5.1 Directories

5.1.1 Format of Directory Entries

There are no requirements for conforming implementations in this subclause.

5.1.2 Directory Operations

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

5.1.2.1 Synopsis

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

5.1.2.2 Description

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

5.1.2.3 Returns

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

5.1.2.4 Errors

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

5.2 Working Directory

5.2.1 Change Current Working Directory

Function: *chdir*()

5.2.1.1 Synopsis

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

5.2.1.2 Description

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

5.2.1.3 Returns

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

5.2.1.4 Errors

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

5.2.2 Get Working Directory Pathname

Function: *getcwd*()

5.2.2.1 Synopsis

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

5.2.2.2 Description

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

5.2.2.3 Returns

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

5.2.2.4 Errors

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

5.3 General File Creation

5.3.1 Open a File

Function: *open*()

5.3.1.1 Synopsis

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

5.3.1.2 Description

NOTE: The General Assertion *GA_syncIODataIntegrityWrite* is defined in §2.2.2.119. It is tested in *write*(), *aio_write*(), and *lio_listio*() instead of in *open*() because they are more appropriate places to test it. The General Assertion *GA_syncIODataIntegrityRead* is defined in §2.2.2.119. It is tested in *real*(), *aio_read*(), and *lio_listio*() instead of in *open*() because they are more appropriate places to test it. The General Assertion *GA_syncIOFileIntegrityRead* is defined in §2.2.1.120. It is tested in *read*(), *aio_read*(), and *lio-listio*() instead of in *open*() because they are more appropriate places to test it. The General Assertion *GA_syncIODataIntegritywrite* is defined in §2.2.2.120. It is tested in *write*(), *aio_write*(), and *listio*() instead of in *open*() because they are more appropriate places to test it.

Also, the O_SYSYNC, O_RSYNC, and O_SYNCH constants are tested for existence and value in §6.5.1.1.

GA_syncOpenWrite

FOR: *write*(), *aio_write*(), and *lio-listio*()

IF {_POSIX_SYNCHRONIZED_IO} **THEN**

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

TEST: The file behaves as if only the O_SYNC flag was set.

TR: Test for regular files.

NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for open: PASS, NO_TEST_NO OPTION

5.3.1.3 Returns

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

5.3.1.4 Errors

1 **IF** {_POSIX_SYNCHRONIZED_IO} **THEN**
IF *PCTS_NO_SYNC_IO_FILE* **THEN**
SETUP: Call *open*()(*path*, *oflag* | *oflag1*).
TEST: The *open*() returns -1 and sets *errno* to [EINVAL].
NOTE: This implementation does not support synchronized I/O for this file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for open: PASS, NO_TEST_SUPPORT, NO_OPTION

5.3.1.5 Errors

2 **IF** {_POSIX_SYNCHRONIZED_IO} **THEN**
IF *PCTS_NO_SYNC_IO_FILE* **THEN**
SETUP: Call *open*()(*path*, *oflag* | *oflag1*).
TEST: The *open*() returns -1 and sets *errno* to [EINVAL].
NOTE: This implementation does not support synchronized I/O for this file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for open: PASS, NO_TEST_SUPPORT, NO_OPTION

5.3.2 Create New File or Rewrite an Existing One

Function: *creat*()

5.3.2.1 Synopsis

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

5.3.2.2 Description

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

5.3.3 Set File Creation Mask

Function: *umask*()

5.3.3.1 Synopsis

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

5.3.3.2 Description

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

5.3.3.3 Returns

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

5.3.3.4 Errors

There are no requirements for conforming implementations in this subclause.

5.3.4 Link to a File

Function: *link*()

5.3.4.1 Synopsis

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

5.3.4.2 Description

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

5.3.4.3 Returns

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

5.3.4.4 Errors

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

5.4 Special File Creation

5.4.1 Make a Directory

Function: *mkdir*()

5.4.1.1 Synopsis

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

5.4.1.2 Description

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

5.4.1.3 Returns

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

5.4.1.4 Errors

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

5.4.2 Make a FIFO Special File

Function: *mkfifo*()

5.4.2.1 Synopsis

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

5.4.2.2 Description

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

5.4.2.3 Returns

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

5.4.2.4 Errors

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

5.5 File Removal

5.5.1 Remove Directory Entries

Function: *unlink*()

5.5.1.1 Synopsis

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

5.5.1.2 Description

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

5.5.1.3 Returns

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

5.5.1.4 Errors

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

5.5.2 Remove a Directory

Function: *rmdir*()

5.5.2.1 Synopsis

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

5.5.2.2 Description

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

5.5.2.3 Returns

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

5.5.2.4 Errors

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

5.5.3 Rename a File

Function: *rename*()

5.5.3.1 Synopsis

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

5.5.3.2 Description

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

5.5.3.3 Returns

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

5.5.3.4 Errors

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

5.6 File Characteristics

5.6.1 File Characteristics: Header and Data Structure

`<sys/stat.h>`

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

5.6.1.1 `<sys/stat.h>` File Types

D_1 If a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether message queues, semaphores, or shared memory objects are implemented as distinct file types does so in §5.6.1.1.
ELSE *NO_OPTION*
Conformance for stat.h: PASS, NO_OPTION

1 **SETUP:** Include the header `<sys/stat.h>`.
TEST: The macros *S_TYPEISMQ*(), *S_TYPEISSEM*(), and *S_TYPEISSHM*() are defined.
Conformance for stat.h: PASS

2 **IF** *PCTS_MQ_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Also, create a message queue and put its information into the *stat* structure referenced by *buf* parameter.
TEST: The macro call *S_TYPEISMQ(buf)* evaluates to a nonzero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

3 **IF** *PCTS_MQ_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a message queue type file into the *stat* structure referenced by *buf* parameter.
TEST: The macro call *S_TYPEISMQ(buf)* evaluates to a zero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

4 **IF** Not *PCTS_MQ_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a message queue type file into the *stat* structure referenced by *buf* parameter.
TEST: The macro call *S_TYPEISMQ(buf)* evaluates to a zero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

5 **IF** *PCTS_SEM_IS_FD* **THEN**
SETUP: Include the header `<sys/stat.h>`. Also, create a semaphore and put its information into the *stat* structure referenced by *buf* parameter.
TEST: The macro call *S_TYPEISSEM(buf)* evaluates to a nonzero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

6 **IF** *PCTS_SEM_IS_FD* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a semaphore type file into the *stat* structure referenced by *buf* parameter.
TEST: The macro call *S_TYPEISSEM(buf)* evaluates to a zero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

7 **IF** *PCTS_SEM_IS_FD* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a semaphore type file into the *stat* structure referenced by *buf* parameter.
TEST: The macro call *S_TYPEISSEM(buf)* evaluates to a zero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

8 **IF** *PCTS_SHM_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Also, create a shared memory object and put its information into the *stat* structure referenced by *buf* parameter.
TEST: The macro call *S_TYPEISSHM(buf)* evaluates to a nonzero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

9 **IF** *PCTS_SHM_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a shared memory object type file into the *stat* structure referenced by *buf* parameter.
TEST: The macro call *S_TYPEISSHM(buf)* evaluates to a zero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

10 **IF** *PCTS_SHM_AS_FILE_TYPE* **THEN**
SETUP: Include the header `<sys/stat.h>`. Do not put the information associated with a shared memory object type file into the *stat* structure referenced by *buf* parameter.
TEST: The macro call *S_TYPEISSHM(buf)* evaluates to a nonzero value.
ELSE *NO_TEST_SUPPORT*
Conformance for stat.h: PASS, NO_TEST_SUPPORT

5.6.1.2 `<sys/stat.h>` File Modes

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

5.6.1.3 `<sys/stat.h>` Time Entries

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

5.6.2 Get File Status

Functions: *stat*(), *fstat*()

5.6.2.1 Synopsis

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

5.6.2.2 Description

1 **SETUP:** Open a shared memory object using *shm_open*(). Call *fstat*() with the *fildes* parameter referring to that shared memory object.
TEST: The *stat* structure pointed to by the *buf* argument has theS_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits updated correctly.
Conformance for fstat: PASS

5.6.2.3 Returns

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

5.6.2.4 Errors

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

5.6.3 Check File Accessibility

Function: *access*()

5.6.3.1 Synopsis

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

5.6.3.2 Description

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

5.6.3.3 Returns

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

5.6.3.4 Errors

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

5.6.4 Change File Modes

Function: *chmod*(), *fchmod* ()

5.6.4.1 Synopsis

1
M_GA_stdC_proto_decl(int;fchmod;,intfildes, mode_t mode; sys/stat.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for fchmod: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(fchmod; sys/stat.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for fchmod: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int;fchmod; sys/stat.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for fchmod: PASS, NO_OPTION

4

M_GA_macro_args(fchmod; sys/stat.h;;;)
SEE: Assertion GA_macro_macro_args in §2.7.3
Conformance for fchmod: PASS, NO_OPTION

5.6.4.2 Description

5 **IF** *PCTSfchmod* **THEN**
SETUP: Open a file where the effective user ID of the calling process matches the file owner.
TEST: A call *fchmod(fildes, mode*) sets the file permission bits from the corresponding bits in the *mode* argument.
TR: For regular files: test all file permission bits.

For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_OPTION

6 **IF** *PCTS_fchmod* **THEN**
IF *PCTS_GAP_MODES_fchmod* **THEN**
SETUP: Open a file where the effective user ID of the calling process does not match the file owner and acquire the appropriate privilege to change the file permission bits of the file using *fchmod*().
TEST: A call *fchmod(fildes, mode)* sets the file permission bits from the corresponding bits in the *mode* argument.
TR : For regular files: test all file permission bits.

For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fchmod:PASS, NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_fchmod* and *PCTS_CHMOD_SUID* **THEN**
SETUP: Open a file where the effective user ID of the calling process matches the file owner.
TEST: A call *fchmod(fildes, mode)* sets the S_ISUID bit from the corresponding bit in the *mode* argument.
TR: For regular files: test all file permission bits.

For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.
ELSE *NO_OPTION*
Conformance for fchmod:PASS, NO_OPTION

8 **IF** *PCTS_fchmod* and *PCTS_CHMOD_SUID* **THEN**
IF *PCTS_GAP_SUID_fchmod* **THEN**
SETUP: Open a file where the effective user ID of the calling process does not match the file owner and acquire the appropriate privilege to change the file permission bits of the file using *fchmod*().

TEST: A call *fchmod(fildes, mode)* sets the S_ISUID bit from the corresponding bit in the *mode* argument.
TR: For regular files: test all file permission bits.

For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fchmod:PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_fchmod* and *PCTS_CHMOD_SGID* **THEN**
SETUP: Open a file where the effective user ID of the calling process matches the file owner.
TEST: A call *fchmod(fildes, mode)* sets the S_ISGID bit from the corresponding bit in the *mode* argument.
TR: For regular files: test all file permission bits.

For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.
ELSE *NO_OPTION*
Conformance for fchmod:PASS, NO_OPTION

10 **IF** *PCTS_fchmod* and *PCTS_CHMOD_SGID* **THEN**
IF *PCTS_GAP_SGID_fchmod* **THEN**
SETUP: Open a file where the effective user ID of the calling process does not match the file owner and acquire the appropriate privilege to change the file permission bits of the file using *fchmod*().
TEST: A call *fchmod(fildes, mode)* sets the S_ISGID bit from the corresponding bit in the *mode* argument.
TR: For regular files: test all file permission bits.

For shared memory objects: test only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fchmod:PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_fchmod* **THEN**
IF *PCTS_RAP_SGID_fchmod* **THEN**
SETUP: Open a regular such that the group ID of the file does not match the effective group ID or one of the supplementary group IDs of the process and one or more of the S_IXUSR, S_IXGRP, or S_IXOTH bits of the file mode are set. Release appropriate privileges to change the S_ISGID bit in the mode of the file.
TEST: Bit S_ISGID (set group ID on execution) in the mode of the file is cleared upon successful return from *fchmod*().
TR: Test for each of the following bits being set individually in the file mode: S_IXUSR, S_IXGRP, and S_IXOTH

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

12 **IF** *PCTS_RAP_SGID_chmod* **THEN**
SETUP: Open a regular such that the group ID of the file does not match the effective group ID or one of the supplementary group IDs of the process and one or more of the S_IXUSR, S_IXGRP, or S_IXOTH bits of the file mode are set. Release appropriate privileges to change the S_ISGID bit in the mode of the file.
TEST: Bit S_ISGID (set group ID on execution) in the mode of the file is cleared upon successful return from *chmod*().
TR: Test for each of the following bits being set individually in the file mode: S_IXUSR, S_IXGRP, and S_IXOTH
ELSE *NO_TEST_SUPPORT*
Conformance for chmod:PASS, NO_TEST_SUPPORT,

13 **IF** *PCTS_fchmod* **THEN**
TEST: Upon successful completion, the *fchmod*() function marks for update the *st_ctime* field of the file.
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_OPTION

5.6.4.3 Returns

R_1 **IF** *PCTS_fchmod* **THEN**
TEST: Upon successful completion, the *fchmod*() function returns a value of zero.
ELSE *NO_OPTION*
SEE: Assertions in §5.6.4.2

R_2 **IF** *PCTS_fchmod* **THEN**
TEST: Upon an unsuccessful completion, the *fchmod*() function returns a value of -1, sets *errno* to indicate the error, and no change to the file *mode* occurs.
ELSE *NO_OPTION*
SEE: Assertions in §5.6.4.4

5.6.4.4 Errors

14 **IF** *PCTS_fchmod* **THEN**
SETUP: Call *fchmod*() with the *fildes* argument not being a valid file descriptor.
TEST: A call to the *fchmod*() function on such a *fildes* returns a value of -1, sets *errno* to [EBADF], and no change to the file *mode* occurs.
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_OPTION

15 **IF** not *PCTS_fchmod* **THEN**
TEST: A call to the *fchmod*() function on such a *fildes* returns a value of -1, sets *errno* to [ENOSYS], and no change to the file *mode* occurs.
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_OPTION

16 **IF** *PCTS_fchmod* **THEN**
SETUP: The effective user ID does not match the owner of the file and the calling process does not have the appropriate privileges.
TEST: A call to the *fchmod*() function on such a *fildes* returns a value of -1, sets *errno* to [EPERM], and no change to the file *mode* occurs.
ELSE *NO_OPTION*
Conformance for fchmod: PASS, NO_OPTION

17 **IF** *PCTS_fchmod* **THEN**
IF *PCTS_ROFS* **THEN**
SETUP: Open a file that resides on a read-only file system.

TEST: A call to the *fchmod*() function on such a *fildes* returns a value of -1, sets *errno* to [EROFS], and no change to the file *mode* occurs.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for fchmod: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_fchmod* and *PCTS_EINVAL_fchmod* **THEN**

SETUP: Open a file such that the *fildes* argument refers to a pipe.

TEST: A call to the *fchmod*() function on such a *fildes* returns a value of -1, sets *errno* to [EINVAL], and no change to the file *mode* occurs.

ELSE *NO_OPTION*

Conformance for fchmod: PASS, NO_OPTION

5.6.5 Change Owner and Group of a File

Function: *chown*()

5.6.5.1 Synopsis

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

5.6.5.2 Description

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

5.6.5.3 Returns

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

5.6.5.4 Errors

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

5.6.6 Set File Access and Modification Times

Function: *utime*()

5.6.6.1 Synopsis

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

5.6.6.2 Description

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

5.6.6.3 Returns

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

5.6.6.4 Errors

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

5.6.7 Truncate a File to a Specified Length

Function: *ftruncate*()

1

M_GA_stdC_proto_decl(int; ftruncate; int fildes, off_t length; unistd.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for ftruncate: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(ftruncate; unistd.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for ftruncate: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; ftruncate; unistd.h;;;)
SEE: Assertion GA_macro__result_decl in §1.3.4
Conformance for ftruncate: PASS, NO_OPTION

4

M_GA_macro_args(ftruncate; unistd.h;;;)
SEE: Assertion GA_macro__args in §2.7.3
Conformance for ftruncate: PASS, NO_OPTION

5.6.7.2 Description

5 **IF** *PCTS_ftruncate* **THEN**
SETUP: Open a regular file where the size of each file exceeds the *length* specified in *ftruncate*() call to be tested.
TEST: The call *ftruncate(fildes, length)* causes the regular file known by the file descriptor *fildes*, to be truncated to *length.*

TR: Test for write-only and read-write access to the file.
ELSE *NO_OPTION*
Conformance for fruncate: PASS, NO_OPTION

D_1 **IF** *PCTS_ftruncate* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the file is changed or its size increased after a call *ftruncate(fildes, length)* if the file previously was smaller than *length* does so in §5.6.7.2.
ELSE *NO_OPTION*
Conformance for fruncate: PASS, NO_OPTION

6 **IF** *PCTS_ftruncate* **THEN**
IF *PCTS_EXTEND_ON_ftruncate* **THEN**
SETUP: Open a regular file whose size is less than the size to which it will be truncated.
TEST: The call *ftruncate(fildes, length)* where length is greater than the size of the file causes the extended area to appear as if it were zero-filled.
TR: Test for write-only and read-write access to the file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fruncate: PASS, NO_TEST_SUPPORT_,NO_OPTION

7 **IF** *PCTS_ftruncate* and *PCTS_shm_open* **THEN**
TEST: A call *ftruncate(fildes, length)* where *fildes* references a shared memory object, sets the size of the shared memory object to *length*.
TR: Test for shared memory objects both larger and smaller than *length.*
read-write access
ELSE *NO_OPTION*
Conformance for fruncate: PASS, NO_OPTION

D_2 **IF** *PCTS_ftruncate* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the result of calling *ftruncate*() on a file that is not a regular file or a shared memory object does so in §5.6.7.2.
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

8 **IF** *PCTS_ftruncate* and *PCTS_mmap* **THEN**
IF *_POSIX_MEMORY_PROTECTION* **THEN**
SETUP: Memory map a file so that the effect of *ftruncate*() is to decrease the size of a file and whole pages beyond the new end were previously mapped.
The whole mapped pages beyond the new end after a call to *ftruncate*() are discarded and references to them result in delivery of a SIGBUS signal.

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

9 **IF** *PCTS_ftruncate* and PCTS_shm_open and *PCTS_mmap* **THEN**
IF *_POSIX_MEMORY_PROTECTION* **THEN**
SETUP: Memory map a shared memory object so that the effect of *ftruncate*() is to decrease the size of the shared memory object and whole pages beyond the new end were previously mapped.
The whole mapped pages beyond the new end after a call to *ftruncate*() are discarded and references to them result in delivery of a SIGBUS signal.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_TEST_SUPPORT, NO_OPTION

10 **IF** *PCTS_ftruncate* **THEN**
TEST: The value of the seek pointer is not to be modified by a ll to *ftruncate*().
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

11 **IF** *PCTS_ftruncate* **THEN**
TEST: Upon successful completion, the *ftruncate*() function marks for update the *st_ctime* and *st_mtime* fields of the file.
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

12 **IF** *PCTS_ftruncate* **THEN**
TEST: The file is unaffected by an unsuccessful call to *ftruncate*().
TR: Test the size of the file as well as the *st_ctime* and *st_mtime* fields of the file.
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

5.6.7.3 Returns

R_1 **IF** *PCTS_ftruncate* **THEN**
TEST: Upon successful completion, the *ftruncate*() function returns to zero.
ELSE *NO_OPTION*
SEE: Assertions in §5.6.7.2.

R_2 **IF** *PCTS_ftruncate* **THEN**
TEST: An unsuccessful call to the *ftruncate*() function returns -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: Assertions in §5.6.7.4.

5.6.7.4 Errors

13 **IF** *PCTS_ftruncate* **THEN**
TEST: A call to *ftruncate*() where the *fildes* argument is not a valid file descriptor open for writing returns -1 and sets *errno* to [EBADF].
TR: Test the size of the file as well as the *st_ctime* and *st_mtime* fields of the file.
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

14 **IF** *PCTS_ftruncate* **THEN**
TEST: A call to *ftruncate*() where the *fildes* argument does not refer to a file on which this operation is possible returns -1 and sets *errno* to [EINVAL]
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

15 **IF** *PCTS_ftruncate* **THEN**
TEST: A call to *ftruncate*() where the file resides on a read-only file system returns -1 and sets *errno* to [EROFS]
ELSE *NO_OPTION*
Conformance for ftruncate: PASS, NO_OPTION

5.7 Configurable Pathname Variables

5.7.1 Get Configurable Pathname Variables

Functions: *pathconf*(), *fpathconf*()

5.7.1.1 Synopsis

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

5.7.1.2 Description

1 **SETUP:** Include the header `<unistd.h>`
TEST: The constants {_PC_ASYNC_IO}, and {_PC_SYNC_IO} are defined.
Conformance for pathconf: PASS

2 **FOR:** *pathconf*(), and *fpathconf*()
IF {_POSIX_ASYNCH_IO} is defined when `<unistd.h>` is included **THEN**
SETUP: Include the header `<unistd.h>`
TEST: A call function() with a name parameter equal to {_POSIX_ASYNCH_IO} returns a value equal to {_POSIX_ASYNCH_IO}.
ELSE *NO_OPTION*
Conformance for pathconf, fpathconf: PASS, NO_OPTION

3 **FOR:** *pathconf*(), and *fpathconf*()
SETUP: Include the header `<unistd.h>`
TEST: A call *function*() with a *name* parameter equal to {_POSIX_ASYNCH_IO} that refers to a file other than a directory returns a value corresponding to the option {_POSIX_ASYNCH_IO} for that file.
Conformance for pathconf, fpathconf: PASS

4 **FOR:** *pathconf*(), and *fpathconf*()
IF {_POSIX_PRIO_IO} is defined when `<unistd.h>` is included **THEN**
SETUP: Include the header `<unistd.h>`
TEST: A call *function*() with a *name* parameter equal to {_POSIX_PRIO_IO} returns a value equal to {_POSIX_PRIO_IO}.
ELSE *NO_OPTION*
Conformance for pathconf, fpathconf: PASS, NO_OPTION

5 **FOR:** *pathconf*(), and *fpathconf*()
SETUP: Include the header `<unistd.h>`
TEST: A call *function*() with a *name* parameter equal to {_POSIX_PRIO_IO} that refers to a file other than a directory returns a value corresponding to the option {_POSIX_PRIO_IO} for that file.
Conformance for pathconf, fpathconf: PASS

6 **FOR:** *pathconf*(), and *fpathconf*()
IF {_POSIX_SYNCH_IO} is defined when `<unistd.h>` included **THEN**
SETUP: Include the header `<unistd.h>`
TEST: A call *function*() with a *name* parameter equal to {_PC_SYNC_IO} returns a value equal to {_PC_SYNC_IO}
ELSE *NO_OPTION*
Conformance for pathconf, fpathconf: PASS, NO_OPTION

7 **FOR:** *pathconf*(), and *fpathconf*()
SETUP: Include the header `<unistd.h>`
TEST: A call *function*() with a *name* parameter equal to {_PC_SYNC_IO} that refers to a file other than a directory returns a value corresponding to the option {_POSIX_SYNC_IO} for that file.
Conformance for pathconf, fpathconf: PASS

D_1 **IF** a PCD.lb documents the following **THEN**
TEST: A PCD.lb that documents whether an implementation supports an association of the variable name with the specified file if the *path* argument to *pathconf*() or *fildes* argument to *pathconf*() refers to a directory does so in §5.7.1.3.
ELSE *NO_OPTION*
Conformance for pathconf, fpathconf: PASS, NO_OPTION

5.7.1.3 Returns

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

5.7.1.4 Errors

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

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

6.1 Pipes

6.1.1 Create an Inter-Process Channel

Function: *pipe()*

6.1.1.1 Synopsis

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

6.1.1.2 Description

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

6.1.1.3 Returns

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

6.1.1.4 Errors

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

6.2 File Descriptor Manipulation

6.2.1 Duplicate an Open File Descriptor

Function: *dup(), dup2()*

6.2.1.1 Synopsis

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

6.2.1.2 Description

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

6.2.1.3 Returns

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

6.2.1.4 Errors

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

6.3 File Descriptor Deassignment

6.3.1 Close a File

Function: *close()*

6.3.1.1 Synopsis

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

6.3.1.2 Description

1 **IF** {_POSIX_ASYNCHRONOUS_IO} **THEN**
TEST: An asynchronous I/O operation that is not canceled completes as if the *close()* operation had not yet occurred.
ELSE *NO_OPTION*
Conformance for close: PASS, NO_TEST, NO_OPTION

2 **IF** {_POSIX_ASYNCHRONOUS_IO}**THEN**
TEST: All operations that are not canceled complete as if the *close()* blocked until the operations completed
ELSE *NO_OPTION*
Conformance for close: PASS, NO_TEST, NO_OPTION

D-1 **TEST:** The PCD.1b documents whether any I/O operations is canceled, and which I/O operation may be canceled upon a call to the *close()* function in §6.3.1.2
Conformance for close: PASS

3 **IF** *PCTS_shm_open* and *PCTS_ mmap* **THEN**
SETUP: Create and map a memory object that remains referenced at the last close.
TEST: After a call to *close()* the entire contents of the memory object persist until the memory object becomes unreferenced.
TR: Test for shared memory.

If {POSIX_MAPPED_FILES}: test for memory mapped files.
ELSE *NO_OPTION*
Conformance for close: PASS, NO_OPTION

4 **IF** *PCTS_shm_open* and *PCTS_mmap* **THEN**
SETUP: Create a memory object and map it in at least two processes. Then unlink the memory object. Perform the last close of the memory object and such that the close results in the memory object becoming unreferenced.
TEST: A call to *close()* removes the memory object.
ELSE *NO_OPTION*
Conformance for close: PASS, NO_OPTION

6.3.1.3 Returns

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

6.3.1.4 Errors

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

6.4 Input and Output

6.4.1 Read from a File

Function: *read()*

6.4.1.1 Synopsis

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

6.4.1.2 Description

1 **IF** *PCTS_read* and {POSIX_SYNCHRONIZED_IO) **THEN**
SETUP: Open a file by calling *open()* with O_RSYNC and O_DSYNC set in the *oflag* parameter.
TEST: A read operation initiated by calling *read()* either completes by transferring an image of the data to the requesting process or, if unsuccessful, by diagnosing and returning an indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityRead in §2.2.2.119
Conformance for read: PASS, NO_TEST, NO_OPTION

2 **IF** *PCTS_read* and {POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by calling *open()* with O_RSYNC and O_DSYNC set in the *oflag* parameter
TEST: At the time that the synchronized read operation initiated by calling *read()* occurs, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data.
TR: Test for regular files.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWbeforeR in §2.2.2.119
Conformance for read: PASS, NO_TEST, NO_OPTION

3 **IF** *PCTS_read* and {POSIX_SYNCHRONIZED_IO) **THEN**
SETUP: Open a file by calling *open()* with O_RSYNC and O_DSYNC set in the *oflag* parameter
TEST: At the time that the synchronized read operation initiated by calling *read()* occurs, any pending write request affecting the data to be read are written to the physical medium containing the file prior to reading the data and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

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

TR: Test for regular files.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIOFileIntegrityRead in §2.2.2.120

Conformance for read: PASS, NO_TEST, NO_OPTION

D-1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of a call to the *read()* function when *fildes* refers to a shared memory object does so in §6.4.1.2**.**
ELSE *NO_OPTION*
Conformance for read: PASS, NO_OPTION

6.4.1.3 Returns

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

6.4.1.4. Errors

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

6.4.2 Write to a File

Function: *write()*

6.4.2.1. Synopsis

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

6.4.2.2. Description

1 **IF** *PCTS_write* and {POSIX_SYNCHRONIZED_IO) **THEN**
SETUP: Open a file by calling *open()* with O_DSYNC set in the *oflag* parameter.
TEST: A write operation initiated by calling *write()* either completes by transferring an image of the data to the physical medium containing the file or, if unsuccessful, by diagnosing and returning an indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWrite in §2.2.2.119
Conformance for write: PASS, NO_TEST, NO_OPTION

2 **IF** *PCTS_write* and {POSIX_SYNCHRONIZED_IO) *PCTS_write* and {POSIX_SYNCHRONIZED_IO) **THEN**
SETUP: Open a file by calling *open()* with O_SYNC set in the *oflag* parameter**.**
TEST: At the time that the synchronized write operation initiated by calling *write()* occurs, the data are written to the physical medium containing the file and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

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

9. Time of last data modification.

10. Time of last file status change.

TR: Test for regular files.

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

ELSE *NO_OPTION*

SEE: Assertion GA_syncIOFileIntegrityWrite in §2.2.2.120

Conformance for write: PASS, NO_TEST, NO_OPTION

D-1 **IF** a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the result of a call to the *write()* function when *fildes* refers to a shared memory object does so in §6.4.2.2

ELSE *NO_OPTION*

Conformance for write: PASS, NO_OPTION

6.4.2.3 Returns

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

6.4.2.4 Errors

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

6.5 Control Operations on Files

6.5.1 Data Definitions for File Control Operations

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

TEST: The constants O_DSYNC, O_RSYNC, and O_SYNC are defined and are unique values with respect to each other and O_CREAT, O_EXCL, O_NOCTTY, O_TRUNC, O_APPEND, O_NONBLOCK, O_RDONLY, O_RDWR, AND O_WRONLY.

Conformance for fcntl.h: PASS

NOTE: IEEE Std 2003.1-1992 {4} was not as explicit as the above assertion regarding the uniqueness of the constants,

6.5.2 File Control

Function: *fcntl()*

6.5.2.1 Synopsis

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

6.5.2.2. Description

All of the assertions for the Description subclause of *fcntl()* except for those relating to file locking and setting file status flags were rewritten from IEEE Std 2003.1-1992 {4} because of the requirement that they apply to shared memory objects and the change in the form of assertions between IEEE Std 2003.1-1992 {4} and this standard.

(IEEE Std 2003.1-1992 {4})04

UNUSED

(IEEE Std 2003.1-1992 {4})05

UNUSED

(IEEE Std 2003.1-1992 {4})06

UNUSED

IEEE Std 2003.1-1992 {4})07
UNUSED

IEEE Std 2003.1-1992 {4})08
UNUSED

IEEE Std 2003.1-1992 {4})09
UNUSED

IEEE Std 2003.1-1992 {4})10
UNUSED

IEEE Std 2003.1-1992 {4})11
UNUSED

IEEE Std 2003.1-1992 {4})12
UNUSED

4 **TEST:** A call *fcntl(fildes, F_DUPFD, arg)* creates a new file descriptor that:

1. Is the lowest numbered one available greater than or equal to the argument *arg*.
2. Refers to the same open file description (file pointer, access mode, and file status flags) as the original file descriptor.
3. Shares the same locks as the original file descriptor, if *fildes* does not refer to a shared memory object.
4. Has the FD_CLOEXEC flag clear.

TR: Test for a regular file. If *PCTS_shm_open:* Test for *fildes* referring to a shared memory object.
Conformance for fcntl: PASS

5 **TEST:** *A* call *fcntl (fildes,F_GETFD)* returns the status of the file descriptor flags.
TR: Test for a regular file. If *PCTS_shm_open*: Test for *fildes* referring to a shared memory object.
Conformance for fcntl: PASS

6 **FOR:** *execl(), execv(), execle(), execve(), execlp()* and *execvp()*
TEST: A call *fcntl (fildes, F_SETFD, arg)* where the FD_CLOEXEC flag in *arg* is nonzero sets the close-on-exec flag for the file associated with *filedes*and *function().*
TR: Test for a regular file. If *PCTS_shm_open:* Test for *fildes* referring to a shared memory object.
Conformance for fcntl: PASS

7 **FOR:** *execl(), execv(), execle(), execve(), execlp(),* and *execvp()*
TEST: A call *fcntl (fildes, F_SETFD, arg)* where the FD_CLOEXEC flag in *arg* is nonzero sets the close-on-exec flag for the file associated with *filedes*and *function().*
TR: Test for a regular file. If *PCTS_shm_open:* Test for *fildes* referring to a shared memory object.
Conformance for fcntl: PASS

8 **TEST:** A call *fcntl(fildes, F_GETFL)* returns the file status flags O_APPEND, O_DSYNC, O_NONBLOCK, O_RSYNC, and O_SYNC and the file access modes O_RDONLY, O_RDWR and O_WRONLY.
TR: Test for a regular file. If *PCTS_shm_open:* Test for *fildes* referring to a shared memory object.
Conformance for fcntl: PASS

9 **TEST:** A call *fcntl(fildes ,F_SETFL, 0)* sets the status flags for the file referenced by *fildes* to 0.
TR: Test for regular files
Conformance for fcntl: PASS

10 **TEST:** A call *fcntl(fildes, F_SETFL, arg)* sets the status flags for the file referenced by *fildes* to values in *arg*.

TR: Test with *arg* having the values O_APPEND, and O_NONBLOCK, individually and together, for regular files.

If {_POSIX_SYNCHRONIZED_IO}: Test with *arg* having the values O_APPEND, O_DSYNC, O_NONBLOCK, O_RSUNC and O_SYNC, individually and all together, on a file supporting {_POSIX_SYNCHRONIZED_IO}.

Conformance for fcntl: PASS

D-1 **IF** *PCTS_shm_open* and PCD.1b that documents the following **THEN**

TEST: A PCD.1b that documents the effect of the values F_SETFL, F_GETLK, F_SETLK, and F_SETLKW. for the argument *cmd* of the function *fcntl()* when the file descriptor *fildes* refers to a shared member object, does so in §6.5.2.2

ELSE *NO_OPTION*

Conformance for fcntl: PASS, NO_OPTION.

6.5.2.3 Returns

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

6.5.2.4 Errors

(IEEE Std 2003.1-1992 {4})41

UNUSED

(IEEE Std 2003.1-1992 {4})42

UNUSED

(IEEE Std 2003.1-1992 {4})48

UNUSED

(IEEE Std 2003.1-1992 {4})49

UNUSED

41 **TEST:** A call *fcntl(fildes, F_DUPFD,arg)* where *arg* is negative returns -1 and sets *errno* to [EINVAL].

TR: Test for a regular file. If PCTS_*shm_open:* Test for *fildes* referring to a shared memory object.

Conformance for fcntl: PASS

42 **IF** a call *sysconf()* with an argument of {_SC_OPEN_MAX} does not return -1

THEN

TEST: A call *fcntl(fcldes, F_DUPFD, arg)* where *arg* is greater than or equal to [OPEN_MAX] returns -1 and sets *errno* to [EINVAL].

TR: Test for a regular file. If PCTS_*shm_open:* Test for *fildes* referring to a shared memory object.

ELSE *NO_TEST_SUPPORT*

Conformance for fcntl: PASS, NO_TEST_SUPPORT

42.1 IF a call *sysconf()* with an argument of {_SC_OPEN_MAX} *returns -1* ***THEN***

TEST: A call *fcntl(fcldes, F_DUPFD, arg)* where *arg* is *equal to* [PCTS_OPEN_MAX] returns a valid file descriptor and does not set *errno* to [EINVAL].

TR: Test for a regular file. If PCTS_*shm_open:* Test for *fildes* referring to a shared memory object.

ELSE *NO_TEST_SUPPORT*

Conformance for fcntl: PASS, NO_TEST_SUPPORT

42.2 IF *PCTS_NO_SYNC_IO_FILE* **THEN**

TEST: A call *fcntl(fcldes, F_SETL, arg***)** where any of the file status flags O_DSYNC, O_RSYNC, OR O_SYNC are set in the *arg* argument returns -1 and sets *errno* to [EINVAL].

TR: Test for a file that does not support synchronized I/O.

ELSE *NO_OPTION*

Conformance for fcntl: PASS, NO_OPTION

48 **IF** {OPEN_MAX}≤ *PCTS_OPEN_MAX* **THEN**
SETUP: Open {OPEN_MAX}*files.*
TEST: A call *fcntl(fildes, F_DUPD, arg)* returns -1 and sets *errno* to [EMFILE].
TR: Test for a regular file. If PCTS_*shm_open:* Test for *fildes* referring to a shared memory object.
ELSE *NO_TEST_SUPPORT*
Conformance for fcntl: PASS, NO_TEST_SUPPORT

48.1 **IF** {OPEN_MAX} > *PCTS_OPEN_MAX* **THEN**
SETUP: Open PCTS_OPEN_MAX files.
TEST: A call *fcntl(fildes, F_DUPD, arg)* returns a valid file descriptor and does not set *errno* to [EMFILE].
TR: Test for a regular file. If PCTS_*shm_open:* Test for *fildes* referring to a shared memory object.
ELSE *NO_TEST_SUPPORT*
Conformance for fcntl: PASS, NO_TEST_SUPPORT

49 **IF** {OPEN_MAX} ≤ *PCTS_OPEN_MAX* **THEN**
SETUP: Open {OPEN_MAX} files then make a file descriptor available whose value will be less than that specified in the *arg* support to *fcntl()*.
TEST: A call *fcntl(fildes, F_DUPD, arg)* where *arg* is greater than the lowest available file descriptor returns -1 and sets *errno* to [EMFILE].
TR: Test for a regular file. If PCTS_*shm_open:* Test for *fildes* referring to a shared memory object.
ELSE *NO_TEST_SUPPORT*
Conformance for fcntl: PASS, NO_TEST_SUPPORT

49.1 **IF** {OPEN_MAX} > *PCTS_OPEN_MAX* **THEN**
SETUP: Open *PCTS_OPEN_MAX* files then make a file descriptor available whose value will be less than that specified in the *arg* argument to *fcntl()*.
TEST: A call *fcntl(fildes, F_DUPD, arg)* returns a valid file descriptor greater than or equal to the *arg* argument and does not set *errno* to [EMFILE].
TR: Test for a regular file. If PCTS_*shm_open:* Test for *fildes* referring to a shared memory object.
ELSE *NO_TEST_SUPPORT*
Conformance for fcntl: PASS, NO_TEST_SUPPORT

6.5.3 Reposition Read/Write File Offset

Function: *lseek()*

6.5.3.1 Synopsis

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

6.5.3.2 Description

D-1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of calling the *lseek()* function when *fildes* refers to a shared memory object does so in §6.5.3.2
ELSE *NO_OPTION*
Conformance for lseek: PASS, NO_OPTION

6.5.3.3. Returns

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

6.5.3.4. Errors

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

6.6 File Synchronization

D-1 **TEST:** The PCD.1b documents the hardware characteristics upon which the implementation relies to assure that data is successfully transferred for synchronized I/O operation in §6.6
Conformance for syncio: PASS

6.6.1 Synchronize the State of a File

Function: *fsync()*

6.6.1.1 Synopsis

1
M_GA_stdC_proto_decl(int;fsync; intfildes; unistd.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for fsync: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(fsync; unistd.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for fsync: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; fsync; unistd.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for fsync: PASS, NO_OPTION

4
M_GA_macro_args (fsync; unistd.h;;;)
SEE Assertion GA_macro_args in §2.7.3
Conformance for fsync: PASS, NO_OPTION

6.6.1.2 Description

D-1 **IF** PCTS_*fsync* **THEN**
TEST: The PCD.1b documents the manner in which the *fsync()* function transfers all data to the storage device associated with the open file description specified by the *fildes* argument in §6.6.1.2
ELSE *NO_OPTION*

5 **IF** PCTS_*fsync* **THEN**
SETUP: Write data to a file so that an error condition will not exist when *fsync()* is called.
TEST: The *fsync()* function does not return until the system has completed transferring all data to the storage device associated with the open file description specified by the *fildes* argument.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fsync: PASS, NO_TEST, NO_OPTION

R-1 **IF** PCTS_*fsync* **THEN**
TEST: A call to the *fsync()* function returns an error that is detected.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertions in §6.6.1.4

D-2 **IF** PCTS_*fsync* **THEN**

TEST: The PCD.1b documents sufficient information for the user to determine whether it is possible to configure an application and installation to ensure that the data is stored with the degree of required stability for the intended use of the *fsync()* function in §6.6.1.2.

ELSE *NO_OPTION*

Conformance for fsync: PASS, NO_OPTION

6 **IF** PCTS_*fsync* and {POSIX_SYNCHRONIZED_IO} **THEN**

SETUP: Open a file for read-write access without specifying the O_DSYNC, O_RSYNC, or O_DSYNC in the *oflags* argument of *open()*. Then write some data to the file and read from another place in the file.

TEST: A call to the *fsync()* function forces all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O file integrity completion state. That is any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

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

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

ELSE *NO_OPTION*

Conformance for fsync: PASS, NO_TEST, NO_OPTION

7 **IF** PCTS_*fsync* and {POSIX_SYNCHRONIZED_IO} **THEN**

SETUP: Open a file for read-write access without specifying the O_DSYNC, O_RSYNC, or O_DSYNC in the *oflags* argument of *open()*. Then write some data to the file..

TEST: A call to the *fsync()* function forces all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O file integrity completion state. That is the data are written to the physical medium containing the file and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode.
2. File serial number.
3. ID of device containing this file.
4. Number of links.
5. User ID of the owner of the file.

6. Group ID of the group of the file.

7. The file size in bytes.

8. Time of last access.

9. Time of last data modification.

10. Time of last file status change.

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

ELSE *NO_OPTION*

Conformance for fsync: PASS, NO_TEST, NO_OPTION

6.6.1.3 Returns

R-2 **IF** PCTS_*fsync* **THEN**

TEST: Upon successful completion, the *fsync()* function returns zero.

ELSE *NO_OPTION*

SEE: Assertions in §6.6.1.2

8 **TEST:** Upon unsuccessful completion, the *fsync()* returns -1 and set *errno* to indicate the error.

SEE: Assertions in §6.6.1.4

Conformance for fsync: PASS

6.6.1.4 Errors

9 **IF** PCTS_*fsync* **THEN**

SETUP: Call the *fsync()* function with the *fildes* argument not being a valid file descriptor.

TEST: *fsync()* returns -1 and set *errno* to [EBADF].

ELSE *NO_OPTION*

Conformance for fsync: PASS, NO_OPTION

10 **IF** PCTS_*fsync* **THEN**

IF PCTS_NO_SYNC_IO_FILE **THEN**

SETUP: Call the *fsync()* function with the *fildes* argument pointing to a file that does not support synchronized I/O

TEST: *fsync()* returns -1 and set *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** not PCTS_*fsync* **THEN**

SETUP: Call the *fsync()* function even though it is not supported by this implementation

TEST: *fsync()* returns -1 and set *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for fsync: PASS, NO_OPTION

12 **IF** PCTS_*fsync* **THEN**

TEST: In the event that any of the queued I/O operations fail, *fsync()* returns the error conditions defined for *read()* and *write().*

ELSE *NO_OPTION*

Conformance for fsync: PASS, NO_TEST, NO_OPTION

6.6.2 Synchronize the Data of a File

Function: *fdatasync*()

6.6.2.1 Synopsis

1

M_GA_stdC_Proto_decl(int; fdatasync: int fildes; unistd.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for fdatasync: PASS[1, 2] NO_OPTION

2

M_GA_commonC_int_result_decl(int; fdatasync: int fildes; unistd.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for fdatasync: PASS[1, 2] NO_OPTION

3

M_GA_macro_result_decl(int; fdatasync: int fildes; unistd.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for fdatasync: PASS[1, 2] NO_OPTION

4

M_GA_macro_args(int; fdatasync: int fildes; unistd.h;;;)
SEE: Assertion GA_macro_result_decl in §2.7.3
Conformance for fdatasync: PASS[1, 2] NO_OPTION

6.6.2.2 Description

5 **IF** PCTS_*fdatasync* **THEN**
SETUP: Write data to a file so that an error condition will not exist when *fdatasync()* is called.
TEST: The *fdatasync()* function does not return until the system has completed transferring all data to the storage device associated with the open file description specified by the *fildes* argument.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_TEST, NO_OPTION

R_1 IF PCTS_*fdatasync* **THEN**
TEST: A call to the *fdatasync()* function returns an error that is detected.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertions in §6.6.2.4

6 **IF** PCTS_*fdatasync* **THEN**
SETUP: Open a file for read-write access without specifying O_DSYNC, O_RSYNC, or O_DSYNC in the *oflags* argument of *open()*. Then write some data to the file and read from another place in the file.
TEST: A call to the *fdatasync()* function forces all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O data integrity completion state. That is it completes by transferring an image of the data to the requesting process and any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_TEST, NO_OPTION

7 **IF** PCTS_*fdatasync* **THEN**
SETUP: Open a file for read-write access without specifying O_DSYNC, O_RSYNC, or O_DSYNC in the *oflags* argument of *open()*. Then write some data to the file.
TEST: A call to the *fdatasync()* function forces all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O data integrity completion state. That is the data are written to the physical medium containing the file.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*

Conformance for fdatasync: PASS, NO_TEST, NO_OPTION

6.6.2.3 Returns

R_2 IF PCTS_*fdatasync* **THEN**
TEST: Upon successful completion, the *fdatasync()* function returns zero.
ELSE *NO_OPTION*
SEE: Assertions in §6.6.2.2

8 **TEST:** Upon unsuccessful completion, the *fdatasync()* returns -1 and set *errno* to indicate the error.
SEE: Assertions in §6.6.2.4
Conformance for fdatasync: PASS

6.6.2.4 Errors

9 **IF** PCTS_*fdatasync* **THEN**
SETUP: Call the *fdatasync()* function with the *fildes* argument not being a valid file descriptor.
TEST: *fdatasync*() returns -1 and set *errno* to [EBADF].
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_OPTION

10 **IF** PCTS_*fdatasync* **THEN**
IF PCTS_NO_SYNC_IO_FILE **THEN**
SETUP: Call the *fdatasync()* function with the *fildes* argument pointing to a file that does not support synchronized I/O.
TEST: *fdatasync*() returns -1 and set *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_OPTION

11 **IF** not PCTS_*fdatasync* **THEN**
SETUP: Call the *fdatasync()* function even though it is not supported by this implementation
TEST: *fdatasync*() returns -1 and set *errno* to [ENOSYS].
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_OPTION

12 **IF** PCTS_*fdatasync* **THEN**
TEST: In the event that any of the queued I/O operations fail, *fdatasync()* returns the error conditions defined for *read()* and *write()*.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for fdatasync: PASS, NO_TEST, NO_OPTION

6.7 Asynchronous Input and Output

6.7.1 Data Definitions for Asynchronous Input and Output

There are no requirements for conforming implementations in this subclause.

6.7.1.1 Asynchronous I/O Control Block

1 **SETUP:** Include the header `<aio.h>`
TEST: An asynchronous I/O control block structure *aiocb* is defined and has at least the following members with the indicated types:

Member Type	Member Name	Description
int	*aio_fildes*	File descriptor.
off_t	*aio_offset*	File offset.
*volatile void**	*aio_buf*	Location of buffer.
size_t	*aio_nbytes*	Length of transfer.
int	*aio_reqprio*	Request priority offset.
struct sigevent	*aio_sigevent*	Signal number and value.
int	*aio_lio_opcode*	Operation to be performed.

Conformance for aio.h: PASS

D-1 **TEST:** The PCD.1b documents any added extensions as permitted in 1.3.1.1 item (2) made to the a*aioch*, 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

Conformance for aio.h: PASS

2 **TEST:** Added extensions to the *aioch* structure, are enabled as required by 1.3.1.1 that is, a Strictly Conforming π1 Application does not need to be modified to execute in such a modified environment.

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

Conformance for aio.h: PASS, NO_TEST

3 **FOR:** *aio_read(), aio_write(),* and *aio_lio_listio()*

IF *PCTS_function* **THEN**

SETUP: Open a file so that O_APPEND is not set for the file descriptor *aio_fildes*, and *aio_fildes* is associated with a device that is capable of seeking

TEST: The operation requested by *function()* takes place at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to the operation with an *offset* argument equal to *aio_offset* and a *whence* argument equal to SEEK_SET.

ELSE *NO_OPTION*

Conformance for aio.h: PASS, NO_OPTION

4 **FOR:** *aio_write(),* and *aio_lio_listio()*

IF *PCTS_function* **THEN**

IF *PCTS_APPEND_WRITE_SAME_ORDER* **THEN**

SETUP: Open a file so that O_APPEND is set for the file descriptor. Open another file where *aio_fildes* is associated with a device that is incapable of seeking.

TEST: Write operations for the *function()* function append to the file in the same order as the calls were made.

TR: Test for both files in the SETUP. Be sure to use the same priority for all writes

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio.h: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 TEST: The PCD.1b documents 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

Conformance for aio.h: PASS

5 **FOR:** *aio_read(), aio_write(),* and *aio_lio_listio()*

IF PCTS_*function* and {_POSIX_PRIORITIZED_IO} and {_POSIX_PRIORITY_SCHEDULING} **THEN**

TEST: Asynchronous I/O is queued in priority order, with the priority of each asynchronous operation based on the current scheduling priority of the calling process.

ELSE *NO_OPTION*

Conformance for aio.h: PASS, NO_OPTION

6 **FOR:** *aio_read(), aio_write(),* and *aio_lio_listio()*
IF *PCTS_function* and {_POSIX_PRIORITIZED_IO}
and {_POSIX_PRIORITY_SCHEDULING} **THEN**
TEST: The *aio_reqprio* member can be used to lower, but not raise, the asynchronous I/O operation priority when it is within the range zero through {AIO_PRIO_DELTA_MAX}, inclusive.
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_OPTION

7 **FOR:** *aio_read(), aio_write()* and *aio_lio_listio()*
IF *PCTS_function* and {POSIX_PRIORITY_SCHEDULING} **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: The priority of an asynchronous request is computed as process scheduling priority minus *aio_reqprio.*.
TR: Test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **FOR:** *aio_read() aio_write,* and *aio_lio_listio()*
IF *PCTS_function* and {POSIX_PRIORITIZED_IO} **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: Requests issued by *function()* with the same priority to a character special file are processed by the underlying device in FIFO order..
TR: Test for both files in the SETUP. Be sure to use the same priority for all writes
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_TEST_SUPPORT, NO_OPTION

D_3 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the order of processing of asynchronous I/O requests of the same priority issued to files that are not character special files does so in §6.7.1.1
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_OPTION

9 **FOR:** *aio_read() aio_write,* and *aio_lio_listio()*
IF *PCTS_function* and {POSIX_PRIORITIZED_IO} and
{POSIX_PRIORITY_SCHEDULING} **THEN**
TEST: The value of *aio_reqprio* has no effect on process scheduling priority.
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_OPTION

10 **FOR:** *aio_read() aio_write,* and *aio_lio_listio()*
IF *PCTS_function* and {POSIX_PRIORITIZED_IO} and
{POSIX_PRIORITY_SCHEDULING} **THEN**
SETUP: Create prioritized asynchronous I/O requests to the same file that are blocked waiting for a resource required for that I/O operation.
TEST: The higher-priority asynchronous I/O requests are granted the resource before lower-priority I/O requests are granted the resource.
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_OPTION

D_4 **TEST:** The PCD.1b documents the relative priority of asynchronous I/O and synchronous I/O in §6.7.1.1.
Conformance for aio.h: PASS

11 **IF** {_POSIX_PRIORITIZED_IO} **THEN**
TEST: The PCD.1b documents for which files I/O prioritization is supported in §6.7.1.1.
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_OPTION

12 **FOR:** *aio_read*(), *aio_write*(), and *aio_lio_listio*()
IF *PCTS_function* **THEN**
SETUP: Use an *aiocb* where the *aio_sigevent.sigev_notify* member is SIGEV_NONE
TEST: After a call to the *function*() function, no signal is posted upon I/O completion, and the error status for the operation and the return status for the operation are set appropriately.
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_OPTION

13 **FOR:** *aio_read*(), *aio_write*(), and *aio_lio_listio*()
IF *PCTS_function* **THEN**
SETUP: Use an *aiocb* where the *aio_sigevent.sigev_notify* member is SIGEV_SIGNAL
TEST: After a call to the *function*() function and upon I/O completion, the signal specified in *aio_sigevent.sigev_signo* is sent to the process.
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_OPTION

14 **FOR:** *aio_read*(), *aio_write*(), and *aio_lio_listio*()
IF *PCTS_function* {_POSIX_REALTIME_SIGNALS} **THEN**
SETUP: Use an *aiocb* where the *aio_sigevent.sigev_notify* member is SIGEV_SIGNAL. Set the signal number to be queued to be in the range SIGRTMIN to SIGRTMAX and the set the SA_SIGINFO flag for that signal number.
TEST: After a call to the *function*() function and upon I/O completion, the signal will be queued to the process and the value specified in *aio_sigevent.sigev_value* will be the *si_value* component of the generated signal.
ELSE *NO_OPTION*
Conformance for aio.h: PASS, NO_OPTION

NOTE: The following requirements of POSIX.1b {3} are explicitly required by the relevant functions and thus are not considered General Assertions and are to be tested in the functions with the requirements:

The return status of the asynchronous operation is the number of bytes transferred by the i/o operation. If the error status is set to indicate an error completion, then the return status is set to the return value that the corresponding *read*(), *write*(), or *fsynch*() call would have returned. When the error status is not equal to [EINPROGRESS], the return status shall reflect the return status of the corresponding synchronous operation.

6.7.1.2 Manifest Constants

15 **SETUP:** Include the header `<aio.h>`.
TEST: The symbols AIO_CANCELED, AIO_NOTCANCELED, and AIO_ALLDONE are defined and have unique values relative to each other.
Conformance for aio.h: PASS

16 **SETUP:** Include the header `<aio.h>`.
TEST: The symbols LIO_WAIT and LIO_NOWAIT are defined and have unique values relative to each other.
Conformance for aio.h: PASS

17 **SETUP:** Include the header `<aio.h>`.
TEST: The symbols LIO_READ, LIO_WRITE, and LIO_NOP are defined and have unique values relative to each other.
Conformance for aio.h: PASS

6.7.2 Asynchronous Read

Function; *aio_read*()

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 §2.7.3
883 *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 §2.7.3
887 *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 §1.3.4
891 *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 §2.7.3
895 *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->aiofildes* 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*

Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_aio_read* and *PCTS_aio_error* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_read*() function when used as a n argument to *aio_error*() returns the error status of the asynchronous operation while it is proceeding.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST, NO_OPTION

10 **IF** *PCTS_aio_read* and *PCTS_aio_error* **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_read*() function when used as a n argument to *aio_error*() returns the error status of the asynchronous operation while it is proceeding.
TR: Test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST SUPPORT, NO_OPTION

11 **IF** *PCTS_aio_read* and *PCTS_aio_return* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_read*() function when used as a n argument to *aio_return*() returns the return status of the asynchronous operation while it is proceeding.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST, NO_OPTION

12 **IF** *PCTS_aio_read* and *PCTS_aio_return* **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_read*() function when used as a n argument to *aio_return*() returns the return status of the asynchronous operation while it is proceeding.
TR: Test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST SUPPORT, NO_OPTION

R_1 **IF** *PCTS_aio_read* **THEN**
TEST: A call *aio_read*() returns an error condition encountered during queuing without having initiated or queued the request.
ELSE *NO_OPTION*
SEE: Assertions in §6.7.2.4

13 **IF** *PCTS_aio_read* **THEN**
TEST: A call *aio_read*() causes the requested operation to take place at the absolute position in the file as given by *aio_offset*, as if *lseek*() were called immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to SEEK_SET.
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_OPTION

D_1 **IF** *PCTS_aio_read* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the value of the file offset for the file after a successful call to enqueue an asynchronous I/O operation does so in §6.7.2.2.
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_OPTION

D_2 **IF** *PCTS_aio_read* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the behavior of a call to the *aio_read*() when the buffer pointed to by *aiocbpl->aio_buf* or the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O completion does so in §6.7.2.2.

ELSE *NO_OPTION*

Conformance for aio_read: PASS, NO_OPTION

D_3 IF *PCTS_aio_read* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the results of simultaneous asynchronous operations using the same *aiocbp* does so in §6.7.2.2.

ELSE NO_OPTION

Conformance for aio_read: PASS, NO_OPTION

15 IF *PCTS_aio_read* and {_POSIX_SYNCHRONIZED_IO} **THEN**

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

TEST: A read operation initiated by calling *aio_read*() either completes by transferring an image of the data to the requesting process or, if unsuccessful, by diagnosing and returning and indicator of the error.

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

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

ELSE *NO_OPTION*

SEE: Assertion GA_syncIODataIntegrityRead in §2.2.2.119

Conformance for aio_read: PASS, NO_TEST, NO_OPTION

16 IF *PCTS_aio_read* and {_POSIX_SYNCHRONIZED_IO} **THEN**

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

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

TR: Test for regular files.

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

ELSE *NO_OPTION*

SEE: Assertion GA_syncIODataIntegrityWbeforeR in §2.2.2.119

Conformance for aio_read: PASS, NO_TEST, NO_OPTION

17 IF *PCTS_aio_read* and {_POSIX_SYNCHRONIZED_IO} **THEN**

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

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

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

10. Time of last file status change.

TR: Test for regular files.

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

ELSE *NO_OPTION*

SEE: Assertion GA_syncIOFileIntegrityRead in §2.2.2.120

Conformance for aio_read: PASS, NO_TEST, NO_OPTION

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

TEST: A PCD.1b that documents the result of any system action that changes the process memory space while an asynchronous I/O is outstanding to the address range being changed does so in §6.7.2.2.

ELSE *NO_OPTION*

Conformance for aio_read: PASS, NO_OPTION

6.7.2.3 Returns

R_2 IF *PCTS_aio read* **THEN**

TEST: The *aio_read*() function returns the value zero to the calling process after the I/O operation is successfully queued.

ELSE *NO_OPTION*

SEE: Assertions in §6.7.2.2.

R_3 IF *PCTS_aio read* **THEN**

TEST: The *aio_read*() function returns the value -1 to the calling process and sets *errno* to indicate the error when the I/O operation not successfully queued.

ELSE *NO_OPTION*

SEE: Assertions in §6.7.2.4.

6.7.2.4 Errors

18 IF *PCTS_aio read* **THEN**

IF {AIO_MAX}≤ *PCTS_AIO_MAX* **THEN**

SETUP: Queue {AIO_MAX} asynchronous I/O operations before calling *aio_read*().

TEST: A call to the *aio_write*() function returns -1 and sets *errno* to [EAGAIN].

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_read: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

19 IF *PCTS_aio read* **THEN**

IF {AIO_MAX}≤ *PCTS_AIO_MAX* and *PCTS_GTI_DEVICE* **THEN**

SETUP: Queue {AIO_MAX} asynchronous I/O operations before calling *aio_read*().

TEST: A call to the *aio_read*() function returns -1 and sets *errno* to [EAGAIN].

TR: Test for a character special device.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

20 IF *PCTS_aio read* **THEN**

IF {AIO_MAX} *PCTS_AIO_MAX* and *PCTS_GIT_DEVICE* **THEN**

SETUP: Queue *PCTS_AIO_MAX* -1 asynchronous I/O operations before calling *aio_read*().

TEST: A call to the *aio_read*() function returns 0 and does not set *errno* to [EAGAIN].

TR: Test for a character special device.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **IF** not *PCTS_aio read* **THEN**
TEST: A call to the *aio_read*() function returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for aio_read: PASS, N0_OPTION

ebadf1 **IF** *PCTS_aio read* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *aio_read*() function where the *aiocbp->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously and then the *aio_read*() function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

ebadf2 **IF** *PCTS_aio read* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *aio_read*() function where the *aiocbp->aio_fildes* argument is not open for reading, the implementation either detects the condition synchronously and then the *aio_read*() function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and then the return status for the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

einval1 **IF** *PCTS_aio read* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *aio_read*() function where the file offset value implied by *aiocbp->aio_offset* would be invalid, *aiocbp->aio_reqprio* is not a valid value, or *aiocbp->aio_nbytes* is an invalid value, the implementation either detects the condition synchronously and then the *aio_read*() function returns -1 and sets *errno* to [EINVAL] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to [EINVAL].
TR: Test separately for each of the three conditions above.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

26 **IF** *PCTS_aio read* **THEN**
IF *PCTS_aio_cancel* and *PCTS_aio_error* and *PCTS_aio_return* **THEN**
SETUP: Call the *aio_read*() function so that it successfully queues the I/O operation.
TEST: After a read operation is canceled by a call to the *aio_cancel*() function, the return status of the asynchronous operation returned by a call to the *aio_return*() function is -1 and the error status returned by a call to the *aio_error*() function is [EINTR] or [ECANCELED].
TR: Test for a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

R_4 IF *PCTS_aio read* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *aio_read*() function where the *aiocbp->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously and then the *aio_read*() function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the value of the call to *aio_error*() is set to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion ebadf1 in §6.7.2.4.

R_5 IF *PCTS_aio read* **THEN**
IF *PCTS_aio_error* **THEN**
TEST: After a call to the *aio_read*() function where the *aiocbp->aio_fildes* argument is not open for reading, the implementation either detects the condition asynchronously and then the *aio_read*() function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion ebadf1 in §6.7.2.4.

29 **IF** *PCTS_aio read* and *PCTS_aio_cancel* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
SETUP: Call the *aio_read*() function so that it successfully queues the I/O operation.
TEST: A read operation that is canceled after a call to the *aio_cancel*() function and before the I/O completed results in a return status returned by a call to the *aio_return*() function that is -1 and the error status returned by a call to the *aio_error*() function that is [ECANCELED].
TR: Test for test for a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_read: PASS, NO_TEST_SUPPORT, NO_OPTION

R_6 IF *PCTS_aio read* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *aio_read*() function where the file offset value implied by *aiocbp->aio_offset* would be invalid, the implementation detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the value of the call to *aio_error*() is set to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion einval in §6.7.2.4.

6.7.3 Asynchronous Write

Function: *aio_write*()

6.7.3.1 Synopsis

1
M_GA_stdC_proto_decl(int; aio_write; struct aiocb * aiocbp; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for aio_write: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(aio_write; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for aio_write: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; aio_write; aio.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for aio_write: PASS[1, 2], NO_OPTION

4

M_GA_macro_args (aio_write; aio.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for aio_write: PASS, NO_OPTION

6.7.3.2 Description

5 **IF** *PCTS_aio_write* **THEN**
TEST: A call to the *aio_write*() function writes *aiocbp->aio_nbytes* to the file associated with *aiocbp->aio_fildes* from the buffer pointed to by *aiocbp->aio_buf.*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

6 **IF** *PCTS_aio_write* **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: A call to the *aio_write*() function returns when the write request has been initiated or queued to the file or device.
TR: Test for a character special device.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS,NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_aio_write* and {_POSIX_PRIORITIZED_IO} **THEN**
SETUP: Use a file for which prioritized I/O is supported.
TEST: The asynchronous operation caused by the *aio_write*() function is submitted at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio.*
NOTE: There is no known portable test method for this assertion.

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

8 **IF** *PCTS_aio_write* and {_POSIX_PRIORITIZED_IO} **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
SETUP: Use a file for which prioritized I/O is supported.
TEST: The asynchronous operation caused by the *aio_write*() function is submitted at a priority equal to the scheduling priority of the process minus *aiocbp->aio_reqprio.*
TR: Test for a character special file.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS,NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_aio_write* and *PCTS_aio_error* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_write*() function when used as an argument to *aio_error*() returns the error status of the asynchronous operation while it is proceeding.
NOTE: There is no known portable test method for this assertion.

ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST, NO_OPTION

10 **IF** *PCTS_aio_write* and *PCTS_aio_error* **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_write*() function when used as an argument to *aio_error*() returns the error status of the asynchronous operation while it is proceeding.
TR: Test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_aio_write* and *PCTS_aio_return* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_write*() function when used as an argument to *aio_return*() returns the error status of the asynchronous operation while it is proceeding.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST, NO_OPTION

12 **IF** *PCTS_aio_write* and *PCTS_aio_return* **THEN**
IF *PCTS_GTI_DEVICE* **THEN**
TEST: The *aiocbp* value used in a call to the *aio_write*() function when used as an argument to *aio_return*() returns the return status of the asynchronous operation while it is proceeding.
TR: Test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_aio_write* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of a call to the *aio_write*() when the buffer pointed to by *aiocbp->aio_buf* or the control block pointed to by *aiocbp* becomes an illegal process prior to asynchronous I/O completion does so in §6.7.3.2.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

13 **IF** *PCTS_aio_write* **THEN**
SETUP: Use a file where the O_APPEND flag is not set for the file descriptor *aio_fildes*.
TEST: A call *aio_write*() causes the requested operation to take place at the absolute position in the file as given by *aio_offset*, as if *lseek*() were called immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to SEEK_SET.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

14 **IF** *PCTS_aio_write* **THEN**
SETUP: Use a file where the O_APPEND flag is set for the file descriptor.
TEST: Write operations caused by calling the *aio_write*() function apend to the file in the same order as the calls were made.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

D_2 **IF** *PCTS_aio_write* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the value of the file offset for the file after a successful call to enqueue an asynchronous I/O operation does so in §6.7.3.2.
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_aio_write* **THEN**
TEST: The *aiocbp->aio_lio_opcode* field is ignored by *aio_write*().
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

D_3 **IF** *PCTS_aio_write* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the results of simultaneous asynchronous operations using the same *aiocbp* does so in §6.7.3.2.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_OPTION

16 **IF** *PCTS_aio_write* and {_POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by callilng *open*() with O_DSYNC set in the *oflag* parameter.
TEST: A write operation initiated by calling *aio_write*() either completes by transferring an image of the data to the physical medium containing the file or, if unsuccessful, by diagnosing and returning an indicator of the error.
TR: Test for regular files, and, if PCTS_GTI_DEVICE, terminals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWrite in §2.2.2.119
Conformance for aio_write: PASS, NO_TEST, NO_OPTION

17 **IF** *PCTS_aio_write* and {_POSIX_SYNCHRONIZED_IO} **THEN** *PCTS_aio_write* and {_POSIX_SYNCHRONIZED_IO}
SETUP: Open a file by calling *open*() with O_SYNC set in the *oflag* parameter.
TEST: At a time that the synchronized write operation initiated by calling *aio_write*() occurs, the data are written to the physical medium containing the file and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

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

TR: Test for regular files.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIOFileIntegrityWrite in §2.2.2.120
Conformance for aio_write: PASS, NO_TEST, NO_OPTION

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

TEST: A PCD.1b that documents the result of any system action that changes the process memory space while an asynchronous I/O is outstanding to the address range being changed does so in §6.7.3.2.

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_OPTION

6.7.3.3 Returns

R_1 IF *PCTS_aio_write* **THEN**

TEST: The *aio_write*() function returns the value zero to the calling process after the I/O operation is successfully queued.

ELSE *NO_OPTION*

SEE: Assertions in §6.7.3.2.

R_2 IF *PCTS_aio_write* **THEN**

TEST: The *aio_write*() function returns the value of -1 to the calling process and sets *errno* to indicate the error when the I/O operation is not successfully queued.

ELSE *NO_OPTION*

SEE: Assertions in §6.7.3.4.

6.7.3.4 Errors

18 **IF** *PCTS_aio_write* **THEN**

IF {AIO_MAX} $\leq$ *PCTS_AIO_MAX* **THEN**

SETUP: Queue {AIO_MAX} asynchronous I/O operations before calling *aio_write*().

TEST: A call to the *aio_write*() function returns -1 and sets *errno* to [EAGAIN].

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_aio_write* **THEN**

IF {AIO_MAX} $\leq$ *PCTS_AIO_MAX* and **THEN**

SETUP: Queue {AIO_MAX} asynchronous I/O operations before calling *aio_write*().

TEST: A call to the *aio_write* function returns -1 and sets *errno* to [EAGAIN]

TR: Test for a character special device.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

20 **IF** *PCTS_aio_write* **THEN**

IF {AIO_MAX} *PCTS_AIO_MAX* and *PCTS_GTI_DEVICE* **THEN**

SETUP: Queue *PCTS_AIO_MAX* asynchronous I/O operations before calling *aio_write*().

TEST: A call to the *aio_write*() function returns 0 and does not set *errno* to [EAGAIN].

TR: Test for a character special device.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **IF** not *PCTS_aio_write* **THEN**

TEST: A call to the *aio_write*() function returns -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_OPTION

ebadf1 **IF** *PCTS_aio_write* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* THEN

TEST: After a call to the *aio_write*() function where the *aiocbp->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously and then the *aio_write*() function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

ebadf2 **IF** *PCTS_aio_write* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* THEN

TEST: After a call to the *aio_write*() function where the *aiocbp->aio_fildes* argument is not open for writing, the implementation either detects the condition synchronously and then the *aio_write*() function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

einval1 **IF** *PCTS_aio_write* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* THEN

TEST: After a call to the *aio_write*() function where the file offset value implied by *aiocbp->aio_offset* would be invalid, *aiocbp->aio_reqprio* is not a valid value, or *aiocbp->aio_nbytes* is an invalid value, the implementation either detects the condition synchronously and then the *aio_write*() function returns -1 and sets *errno* to [EINVAL] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EINVAL]

TR: Test separately for each of the three conditions above.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

22 **IF** *PCTS_aio_write* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* THEN

SETUP: Call the *aio_write*() function so that it successfully queues the I/O operation for a pipe or FIFO that has the O_NONBLOCK flag set for the file descriptor and with the *aiocbp->aio_nbytes* is less than or equal to {PIPE_BUF} and where there is insufficient capacity to accept the data.

TEST: The return status of the asynchronous operation returned by a call to the *aio_return*() function is -1 and the error status returned by a call to the *aio_error*() function is [EAGAIN].

TR: Test for both a pipe and FIFO. If {PIPE_BUF} is greater than *PCTS_PIPE_BUF*, test with values of *aiocbp->aio_nbytes* up to and including *PCTS_PIPE_BUF*.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

24 **IF** *PCTS_aio_write* **THEN**

IF *PCTS_aio_cancel* and *PCTS_aio_error* and *PCTS_aio_return* THEN

SETUP: Call the *aio_write*() function so that it successfully queues the I/O operation.

TEST: After a write operation is canceled by a call to the *aio_cancel*() function and no data were transferred, the return status of the asynchronous operation returned by a call to the *aio_return*() function is -1 and the error status returned by a call to the *aio_error*() function is [EINTR] or [ECANCELED].

TR: Test for test for a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

25 **IF** *PCTS_aio_write* **THEN**

IF the implementation supports a maximum file size **THEN**

TEST: An attempt to write a file that would exceed an implementation-defined maximum size by calling the *aio_write*() function and where the write operation is successfully queued, causes the error status for the asynchronous operation to be [EFBIG].

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

26 **IF** *PCTS_aio_write* and {_POSIX_JOB_CONTROL} **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Make the testing process part of the background process group, set TOSTOP, ignore or block SIGTTOU signals, and make the process group of the process orphaned. Call the *aio_write*() function so that it successfully queues the I/O operation to write from its controlling terminal.

TEST: The write operation will fail and the return status of the asynchronous operation returned by a call to the *aio_return*() function is -1 and the error status returned by a call to the *aio_error*() function is [EIO].

TR: Test for test for a pipe and a FIFO.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

27 **IF** *PCTS_aio_write* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* and *PCTS_DETECT_ENOSPC* **THEN**

SETUP: Fill a device so that there is no more space available on it for data. Call the *aio_write*() function so that it successfully queues the I/O operation.

TEST: After a call to the *aio_write*() function, the implementation either detects the condition synchronously and the *aio_write* () function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

28 **IF** *PCTS_aio_write* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* THEN

SETUP: Call the *aio_write*() function so that it successfully queues the I/O operation.

TEST: After a call to the *aio_write*() function to write to a pipe or FIFO that is not open for reading for any process, the implementation either detects the condition synchronously and the *aio_write*() function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

R_3 IF *PCTS_aio_write* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *aio_write*() function where the *aiocbp_>aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition asynchronously and the *aio_write*() function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and the error status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion ebadf1 in §6.7.3.4.

R_4 IF *PCTS_aio_write* **THEN**
IF *PCTS_aio_error* **THEN**
TEST: After a call to the *aio_write*() function where the *aiocbp_>aio_fildes* argument is not open for writing, the implementation either detects the condition synchronously and then the *aio_write*() function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion ebadf1 in §6.7.3.4.

29 **IF** *PCTS_aio_write* and *PCTS_aio_cancel* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
SETUP: Call the *aio_write*() function so that it successfully queues the I/O operation.
TEST: A write operation that is canceled after call to the *aio_cancel*() function and before the I/O completed results in a return status returned by a call to the *aio_return*() function that is -1 and the error status returned by a call to the *aio_error*() function that is [ECANCELED].
TR: Test for test for a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_write: PASS, NO_TEST_SUPPORT, NO_OPTION

R_5 IF *PCTS_aio_write* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *aio_write*() function where file offset value implied by *aiocbp->aio_offset* would be invalid, the implementation detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion einval in §6.7.3.4.

6.7.4 List Directed I/O

Function *lio_listio*()

6.7.4.1 Synopsis

1
M_GA_stdC_proto_decl(int; lio_listio; int mode, struct aiocb *const list[], int nent, struct sigevent *sig; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3

Conformance for lio_listio: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(lio_listio; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for lio_listio: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; lio_listio; aio.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for lio_listio: PASS, NO_OPTION

4

M_GA_macro_args (lio_listio; aio.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for lio_listio: PASS, NO_OPTION

6.7.4.2 Description

5 **IF** *PCTS_lio_listio* **THEN**
TEST: A successful call to the *lio_listio*() function, where the *mode* argument is LIO_WAIT, waits until all I/O is complete, ignores the *sig* argument, and returns zero.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

6 **IF** *PCTS_lio_listio* **THEN**
TEST: A successful call to the *lio_listio*() function, where the *mode* argument is LIO_NOWAIT and where the *sig* argument is **NULL** returns immediately, does not deliver a signal upon completion of all I/O operations, and returns zero.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

7 **IF** *PCTS_lio_listio* **THEN**
TEST: A successful call to the *lio_listio*() function, where the *mode* argument is LIO_NOWAIT, and where the *sig* argument is not **NULL** and the *sigev_signo* member of the *sigevent* structure referenced by *sig* is zero, returns immediately, does not deliver a signal upon completion of all I/O operations, and returns zero.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

8 **IF** *PCTS_lio_listio* **THEN**
TEST: A successful call to the *lio_listio*() function, where the *mode* argument is LIO_NOWAIT and where the *sig* argument is not **NULL** and the *sigev_signo* member of the *sigevent* structure referenced by *sig* is not zero, the signal number indicated by *sigev_signo* is delivered when all the requests in *list* have completed and the call returns a value of zero immediately.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

D_1 **IF** *PCTS_lio_listio* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the order in which the I/O requests enumerated by *list* are submitted does so in §6.7.4.2.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

9 **IF** *PCTS_lio_listio* **THEN**
TEST: A successful call to the *lio_listio*() function with **NULL** elements in the *list* argument ignores the **NULL** elements and returns to zero.

TR: Test with **NULL** elements interspersed in the *list*.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

10 **IF** *PCTS_lio_listio* **THEN**
TEST: A successful call to the *lio_listio*() function where the *aio_lio_opcode* field of an *aiocb* structure specifies the LIO_NOP operation causes the list entry to be ignored.
TR: Test with multiple file descriptors and buffers in a single call with LIO_NOP operations.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

lio_read_op
IF *PCTS_lio_listio* **THEN**
TEST: A successful call to the *lio_listio*() function where the *aio_lio_opcode* field of an *aiocb* structure specifies the LIO_READ operation causes an I/O operation to be submitted as if by a call to *aio_read* with the *aiocb* structure.
TR: Test with multiple file descriptors and buffers in a single call. If *POSIX_PRIORITY_SCHEDULING*, test with multiple scheduling priorities.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

lio_write_op
IF *PCTS_lio_listio* **THEN**
TEST: A successful call to the *lio_listio*() function where the *aio_lio_opcode* field of an *aiocb* structure specifies the LIO_WRITE operation causes an I/O operation to be submitted as if by a call to *aio_write* with the *aiocb* equal to the address of the *aiocb* structure.
TR: Test with multiple file descriptors and buffers in a single call. If *POSIX_PRIORITY_SCHEDULING*, test with multiple scheduling priorities.
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

R_1 **IF** *PCTS_lio_listio* **THEN**
TEST: The *list* element further describes the I/O operation to be performed, in a manner identical to that of the corresponding *aiocb* structure when used by the *aio_read*() and *aio_write* () functions.
ELSE *NO_OPTION*
SEE: The lio_read_op and lio_write_op assertions.

11 **IF** *PCTS_lio_listio* and {_POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by calling *open*() with O_RSYNC and O_DSYNC set in the *oflag* parameter.
TEST: A read operation initiated by calling *lio_listio*() either completes by transferring an image of the data to the requesting process, or if unsuccessful, by diagnosing and returning an indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityRead in §2.2.2.119
Conformance for lio_listio: PASS, NO_TEST, NO_OPTION

12 **IF** *PCTS_lio_listio* and {_POSIX_SYNCHRONIZED_IO} **THEN**
SETUP: Open a file by calling *open*() with O_RSYNC and O_SYNC set in the *oflag* parameter.
TEST: At the time that the synchronized read operation initiated by calling *lio_listio*() occurs, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

1. File mode.

2. File serial number.

3. ID of device containing this file.

4. Number of links.

5. User ID of the owner of the file.

6. Group ID of the group of the file.

7. The file size in bytes.

8. Time of last access.

9. Time of last data modification.

10. Time of last file status change.

TR: Test for regular files .

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

ELSE *NO_OPTION*

SEE: Assertion GA_syncIOFileIntegrityRead in §2.2.2.120

Conformance for lio_listio: PASS, NO_TEST, NO_OPTION

14 **IF** *PCTS_lio_listio* and {_POSIX_SYNCHRONIZED_IO} **THEN**

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

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

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

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

ELSE *NO_OPTION*

SEE: Assertion GA_syncIODataIntegrityWrite in §2.2.2.119

Conformance for lio_listio: PASS, NO_TEST, NO_OPTION

15 **IF** *PCTS_lio_listio* and {_POSIX_SYNCHRONIZED_IO} **THEN** *PCTS_lio_listio* and {_POSIX_SYNCHRONIZED_IO}

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

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

1. File mode.

2. File serial number.

3. ID of device containing this file.

4. Number of links.

5. User ID of the owner of the file.

6. Group ID of the group of the file.

7. The file size in bytes.

8. Time of last access.

9. Time of last data modification.

10. Time of last file status change.

TR: Test for regular files.

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

ELSE *NO_OPTION*

SEE: Assertion GA_syncIOFileIntegrityWrite in §2.2.2.120

Conformance for lio_listio: PASS, NO_TEST, NO_OPTION

6.7.4.3 Returns

R_2 IF *PCTS_lio_listio* **THEN**

TEST: A call to the *lio_listio*() function where the *mode* argument has the value LIO_NOWAIT returns the value zero and queues the I/O operations.

ELSE *NO_OPTION*

SEE: Assertions in §6.7.4.2.

R_3 IF *PCTS_lio_listio* **THEN**

TEST: An unsuccessful call to the *lio_listio*() function where the *mode* argument has the value LIO_NOWAIT returns the value -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: Assertions in §6.7.4.4.

R_4 IF *PCTS_lio_listio* **THEN**

TEST: A successful call to the *lio_listio*() function where the *mode* argument has the value LIO_WAIT returns the value zero when all the indicated I/O has completed successfully.

ELSE *NO_OPTION*

SEE: Assertions in §6.7.4.2.

R_5 IF *PCTS_lio_listio* **THEN**

TEST: An unsuccessful call to the *lio_listio*() function where the *mode* argument has the value LIO_WAIT returns the value -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: Assertions in §6.7.4.4.

16 IF *PCTS_lio_listio* **THEN**

TEST: A successful call to the *lio_listio*() function where some individual requests fail does not prevent completion of any other individual request and returns zero.

TR: Intersperse individual I/O operations that will fail with those that will succeed.

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

R_6 IF *PCTS_lio_listio* **THEN**

TEST: The error statuses returned by a call to *lio_listio*() with the *aio_lio_opcode* field of an *aiocb* structure equal to LIO_READ or LIO_WRITE are identical to those returned as the result of an *aio_read*() or *aio_write*() function, respectively.

ELSE *NO_OPTION*

SEE: Assertions starting with lio_read and lio_write in §6.7.4.4.

6.7.4.4 Errors

17 IF *PCTS_lio_listio* **THEN**

SETUP: Queue enough asynchronous I/O operations before calling *lio_listio*() to perform the test so that not all I/O requests in the call will have a resource to be queued.

TEST: A call to the *lio_listio*() function returns -1 and sets *errno* to [EAGAIN].

TR: Test with both LIO_READ and LIO_WRITE operations.

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

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST, NO_OPTION

18 **IF** *PCTS_lio_listio* **THEN**
IF {AIO_MAX}≤ *PCTS_AIO_MAX* **THEN**
SETUP: Queue {AIO_MAX} or fewer asynchronous I/O operations before calling *lio_listio*() to perform the test.
TEST: A call to the *lio_listio*() function returns -1 and sets *errno* to [EAGAIN].
TR: Test with both {AIO_MAX} and fewer queued operations. When testing with less than {AIO_MAX} operations queued, make sure that some, but not all, of the I/O operations in the call can be queued. Test with both LIO_READ and LIO_WRITE operations.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_lio_listio* **THEN**
IF {AIO_MAX}≤ *PCTS_AIO_MAX* and *PCTS_GTI_DEVICE* **THEN**
SETUP: Queue {AIO_MAX} asynchronous I/O operations before calling *lio_listio*() for the test.
TEST: A call to the *lio_listio*() function returns -1 and sets *errno* to [EAGAIN].
TR: Test for a character special device. Test with both {AIO_MAX} and fewer queued operations. When testing with less than {AIO_MAX} operations queued, make sure that some, but not all, of the I/O operations in the call can be queued. Test with both LIO_READ and LIO_WRITE operations.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

20 **IF** *PCTS_lio_listio* **THEN**
IF {AIO_MAX}> PCTS_AIO_MAX and *PCTS_GTI_DEVICE* **THEN**
SETUP: Queue *PCTS_AIO_MAX* - *nent* asynchronous I/O operations before calling *lio_listio*().
TEST: A call to the *lio_listio*() function with *nent* entries returns 0 and does not set *errno* to [EAGAIN].
TR: Test for a character special device. Test with both LIO_READ and LIO_WRITE operations.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

NOTE: Is there anything special about the [EAGAIN] error that depends on *nent*?

lio_read_ebadf1
IF *PCTS_lio_listio* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_READ and where the *list[]->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously and then the *lio_listio*() function returns -1 and sets *errno* to [EIO] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EIO] .
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

lio_read_ebadf2
IF *PCTS_lio_listio* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_READ and *list[]->aio_fildes* argument is not open for reading, the implementation either detects the condition synchronously and then the *lio_listio*() function returns -1 and sets *errno* to [EIO] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EIO] .

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

lio_read_einval1

IF *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_READ and the file offset value implied by *list[]->aio_offset* would be invalid, *list[]->aio_reqprio* is not a valid value, or *list[]->aio_nbytes* is an invalid value, the implementation either detects the condition synchronously and then the *lio_listio*() function returns -1 and sets *errno* to [EIO] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to [EIO] .

TR: Test separately for each of the three conditions above.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

26 **IF** *PCTS_lio_listio* and {_POSIX_JOB_CONTROL} **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Make the testing process part of the background process group and make the process group of the process orphaned. Call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_READ and so that it successfully queues the I/O operation to read from its controlling terminal.

TEST: The read operation will fail and the return status of the asynchronous operation returned by a call to the *aio_return* () function is -1 and the error status returned by a call to the *aio_error*() function is [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

27 **IF** *PCTS_lio_listio* and {_POSIX_JOB_CONTROL} **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Make the testing process part of the background process group and have the process ignore or block the SIGTTIN signal. Call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_READ and so that it successfully queues the I/O operation to read from its controlling terminal.

TEST: The read operation will fail and the return status of the asynchronous operation returned by a call to the *aio_return* () function is -1 and the error status returned by a call to the *aio_error*() function is [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

R_7 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_READ and where the *list[]->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously and

the *lio_listio*() function returns -1 and sets *errno* to [EIO] or it detects the condition asynchronously and the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_read_ebadf1 in §6.7.4.4

R_8 IF *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* **THEN**

TEST: After a call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_READ and where the *list[]->aio_fildes* argument is not open for reading, the implementation either detects the condition synchronously and then the *lio_listio*() function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_read_ebadf2 in §6.7.4.4.

28 IF *PCTS_lio_listio* and *PCTS_aio_cancel* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Call to the *lio_listio*() function with a *list[]->aio_lio_opcode* equal to LIO_READ and so that it successfully queues the I/O operation.

TEST: A read operation that is canceled after a call to the *aio_cancel*() function and before the I/O completed results in a return status returned by a call to the *aio_return*() function that is -1 and the error status returned by a call to the *aio_error*() function that is [ECANCELED].

TR: Test for test for a pipe and a FIFO.

If *PCTS_GTI_DEVICE*,, test for a character special file.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

R_9 IF *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: After a call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_READ and where the file offset value implied by *list[]->aio_offset* would be invalid, the implementation detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EINVAL].

TEST: The read operation will fail and the return status of the asynchronous operation returned by a call to the *aio_return* () function is -1 and the error status returned by a call to the *aio_error*() function is [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_read_einval in §6.7.4.4.

lio_write_ebadf1

IF *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_WRITE and where the *list[]->aio_fildes* argument is not a valid file descriptor, the implementation either detects the condition synchronously and then the *lio_listio*() function returns -1 and sets *errno* to [EIO] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EIO] .

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

lio_write_ebadf2
IF *PCTS_lio_listio* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_WRITE and where the *list[]->aio_fildes* argument that is not open for writing, the implementation either detects the condition synchronously and then the *lio_listio*() function returns -1 and sets *errno* to [EIO] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EIO] .
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

lio_write_einval1
IF *PCTS_lio_listio* **THEN**
IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**
TEST: After a call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_WRITE and file offset value implied by *list[]->aio_offset* would be invalid, the implementation either detects the condition synchronously and then the *lio_listio*() function returns -1 and sets *errno* to [EIO] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to [EIO] .
TR: Test separately for each of the three conditions above.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

If *PCTS_GTI_DEVICE*, test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

31 **IF** *PCTS_lio_listio* **THEN**
IF *PCTS_aio_cancel* and *PCTS_aio_error* and *PCTS_aio_return* **THEN**
SETUP: Call to the *lio_listio*() function with an *list[]->aio_lio_opcode* equal to LIO_WRITE and so that it successfully queues the I/O operation.
TEST: After a write operation is canceled by a call to the *aio_cancel*() function and no data were transferred, the return status of the asynchronous operation returned by a call to the *aio_return*() function is -1 and the error status returned by a call to the *aio_error*() function is [EINTR] or [ECANCELED].
TR: Test for test for a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

32 **IF** *PCTS_lio_listio* **THEN**
IF the implementation supports a maximum file size **THEN**
TEST: An attempt to write a file that would exceed an implementation-defined maximum size by calling the *lio_listio*() function and where the write operation

is successfully queued, causes the error status for the asynchronous operation to be [EFBIG].

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

33 **IF** *PCTS_lio_listio* and {_POSIX_JOB_CONTROL} **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Make the testing process part of the background process group, set TOSTOP, ignore or block SIGTTOU signals, and make the process group of the process orphaned. Call the *lio_listio*() function so that it successfully queues the I/O operation to write from its controlling terminal.

TEST: The write operation will fail and the return status of the asynchronous operation returned by a call to the *aio_return*() function is -1 and the error status returned by a call to the *aio_error*() function is [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

34 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* and *PCTS_DETECT_ENOSPC* **THEN**

SETUP: Fill a device so that there is no more space available on it for data. Call the *lio_listio*() function so that it successfully queues a write I/O operation.

TEST: After a call to the *lio_listio*() function the implementation either detects the condition synchronously and the *lio_listio*() function returns -1 and sets *errno* to [EIO] or it detects the condition asynchronously and the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

35 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Call the *lio_listio*() function so that it successfully queues a write I/O operation.

TEST: After a call to the *lio_listio*() function to write to a pipe or FIFO that is not open for reading by any process, the implementation either detects the condition synchronously and the *lio_listio*() function returns -1 and sets *errno* to [EIO] or it detects the condition asynchronously and the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EIO].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

R_10 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function where the *list[]->aio_fildes* argument is not a valid file descriptor, the implementation detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_write_ebadf1 in §6.7.4.4.

R_11 **IF** *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* **THEN**

TEST: After a call to the *lio_listio*() function where the *list[]->aio_fildes* argument is not open for writing, the implementation detects the condition synchronously and then the *lio_listio*() function returns -1 and sets *errno* to [EBADF] or it detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EBADF].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_write_ebadf2 in §6.7.4.4.

36 **IF** *PCTS_lio_listio* and *PCTS_aio_cancel* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

SETUP: Call the *lio_listio*() function so that it successfully queues a write I/O operation.

TEST: A write function that is canceled after a call to the *aio_cancel*() function and before the I/O completed results in a return status returned by a call to the *aio_return*() function that is -1 and the error status returned by a call to the *aio_error*() function that is [ECANCELED].

TR: Test for test for a pipe and a FIFO.

If *PCTS_GTI_DEVICE*, test for a character special file.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_TEST_SUPPORT, NO_OPTION

R-12 IF *PCTS_lio_listio* **THEN**

IF *PCTS_aio_error* and *PCTS_aio_return* **THEN**

TEST: After a call to the *lio_listio*() function for a write operation where the file offset value implied by *list[]->aio_offset* would be invalid, the implementation detects the condition asynchronously and then the return status of the asynchronous operation is set to -1, and the error status of the asynchronous operation is set to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion lio_write_einval1 in §6.7.4.4.

37 **IF** *PCTS_lio_listio* **THEN**

TEST: A call to the *lio_listio*() where the *mode* argument is not a proper value returns -1 and sets *errno* to [EINVAL].

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

38 **IF** *PCTS_lio_listio* **THEN**

TEST: A call to the *lio_listio*() where the value of *nent* is greater than {AIO_LISTIO_MAX}, returns -1 and sets *errno* to [EINVAL].

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

39 **IF** *PCTS_lio_listio* **THEN**

SETUP: Call to *lio_listio*() where the *mode* argument is LIO_WAIT and there are at least two I/O operations and at least one of which will complete before the others and send a signal indicating its completion.

TEST: Such an *lio_listio*() call will receive a signal while waiting for all I/O requests to complete during an operation and return -1 and set *errno* to [EINTR] and outstanding I/O requests are not canceled.

TR: Test for a signal also generated by another process.

ELSE *NO_OPTION*

Conformance for lio_listio: PASS, NO_OPTION

40 **IF** not *PCTS_lio_listio* **THEN**

TEST: A call to the *lio_listio*() function returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for lio_listio: PASS, NO_OPTION

6.7.5 Retrieve Error Status of Asynchronous I/O Operation

Function: *aio_error*()

6.7.5.1 Synopsis

1
M_GA_stdC_proto_decl(int; aio_error; const struct aiocb * aiocbp; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for aio_error: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(aio_error; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for aio_error: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; aio_error; aio.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for aio_error: PASS, NO_OPTION

4
M_GA_macro_args (aio_error; aio.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for aio_error: PASS, NO_OPTION

6.7.5.2 Description

5 **IF** *PCTS_aio_error* **THEN**
TEST: A call to the *aio_error*() function returns the error status associated with the *aiocb* structure referenced by the *aiocbp* argument.
ELSE *NO_OPTION*
Conformance for aio_error: PASS, NO_OPTION

R_1 IF *PCTS_aio_error* **THEN**
TEST: The error status returned by a call to *aio_error*() is the *errno* value that would be set by the corresponding *read*(), *write*(), or *fsync*() operation.
ELSE *NO_OPTION*
SEE: Assertions for *aio_read*() in §6.7.2.4.

6 **IF** *PCTS_aio_error* **THEN**
TEST: A call to the *aio_error*() function where the *aiocb* structure referenced by the *aiocbp* argument refers to an operation that has not yet completed returns the value [EINPROGRESS].
ELSE *NO_OPTION*
Conformance for aio_error: PASS, NO_OPTION

6.7.5.3 Returns

7 **IF** *PCTS_aio_error* **THEN**
TEST: A call to the *aio_error*() function where the *aiocb* structure referenced by the *aiocbp* argument refers to an asynchronous I/O operation that has completed successfully, returns 0.

ELSE *NO_OPTION*
Conformance for aio_error: PASS, NO_OPTION

R-2 **TEST:** The error status returned by a call to *aio_error*() for an asynchronous I/O operation that has completed unsuccessfully is the *errno* value that would be set by the corresponding *read*(), *write*(), or *fsync*() operation.
ELSE *NO_OPTION*
SEE: Assertions for *aio_read*() in §6.7.2.4.

8 **IF** *PCTS_aio_error* **THEN**
TEST: A call to the *aio_error*() function where the *aiocb* structure referenced by the *aiocbp* argument refers to an asynchronous I/O operation has not yet completed returns [EINPROGRESS].
ELSE *NO_OPTION*
Conformance for aio_error: PASS, NO_OPTION

6.7.5.4 Errors

9 **IF** not *PCTS_aio_error* **THEN**
TEST: A call to the *aio_error*() returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for aio_error: PASS, NO_OPTION

10 **IF** *PCTS_aio_error* **THEN**
IF *PCTS_DETECT_AIO_ERROR_AIOCBP* **THEN**
TEST: A call to the *aio_error*() where the *aiocbp* argument refers to an asynchronous operation whose return status has previously been retrieved returns -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_error: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_aio_error* **THEN**
IF *PCTS_DETECT_AIO_ERROR_AIOCBP* **THEN**
TEST: A call to the *aio_error*() function where the *aiocbp* argument refers to an invalid *aiocb* structure returns -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_error: PASS, NO_OPTION

6.7.6 Retrieve Return Status of Asynchronous I/O Operation

Function: *aio_return*()

6.7.6.1 Synopsis

1
M_GA_stdC_proto_decl(int; aio_return; struct aiocb * aiocbp; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for aio_return: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(ssize_t; aio_return; aio.h;;;)
SEE: Assertion GA_commonC_result_decl in §2.7.3
Conformance for aio_return: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(ssize_t; aio_return; aio.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for aio_return: PASS, NO_OPTION

4
M_GA_macro_args (aio_return; aio.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for aio_return: PASS, NO_OPTION

6.7.6.2 Description

return_status
IF *PCTS_aio_return* THEN
TEST: A call to the *aio_return*() function returns the return status associated with the *aiocb* structure referenced by the *aiocbp* argument.
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

R_1 IF *PCTS_aio_return* **THEN**
TEST: The return status for an asynchronous I/O operation is the value that would be returned by the corresponding *read*(), *write*(), or *fsync*() function call.
ELSE *NO_OPTION*
SEE: Assertions for *aio_read*() in §6.7.2.4.

D_1 IF *PCTS_aio_return* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the return status for an operation returned by a call to *aio_return*() when the error status is equal to [EINPROGRESS] does so in §6.7.6.2.
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

R_2 IF *PCTS_aio_return* **THEN**
SETUP: Initiate an asynchronous I/O operation and let it complete. Retrieve the return status by calling *aio-return*().
TEST: A call to the *aio_return*() or the *aio_error*() function using the same *aiocb* structure as used in a previously successful call to *aio_return*() returns -1 and sets *errno* to [EINVAL].
TR: Test for calling both *aio_return*() and *aio_error*().
ELSE *NO_OPTION*
SEE: Assertion einval1 in §6.7.6.4.

5 IF *PCTS_aio_return* **THEN**
SETUP: Initiate and complete an asynchronous I/O operation. Call the *aio_return*() function to retrieve its return status. Submit the same *aiocb* structure referred to by *aiocbp* in the *aio_return*() call to another asynchronous operation.
TEST: A call to the *aio_return*() function can reuse an *aiocb* structure used in a previously successful *aio_return*() call after it has been used to initiate another asynchronous I/O operation.
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

6.7.6.3 Returns

R_3 IF *PCTS_aio_return* **THEN**
SETUP: After an asynchronous I/O operation has completed, the return status, as described for *read*(), *write*(), and *fsync*() , is returned.
ELSE *NO_OPTION*
SEE: Assertion return_status in §6.7.6.4.

D_2 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents defines the results of a call to the *aio_return*() function when the asynchronous I/O operation has not yet completed does so in §6.7.6.4.
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

6.7.6.4 Errors

einval1
IF *PCTS_aio_return* **THEN**
SETUP: Initiate an asynchronous I/O operation and let it complete. Retrieve the return status by calling *aio_return*().
TEST: A call to the *aio_return*() or the *aio_error*() function using the same *aiocb* structure as used in a previously successful call to *aio_return*() returns -1 and sets *errno* to [EINVAL].
TR: Test for calling *aio_return*(). If *PCTS_DETECT_AIO_ERROR_AIOCBP* then test for *aio_error*().
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

6 **IF** *PCTS_aio_return* **THEN**
TEST: A call to the *aio_return*() function where the *aiocbp* argument refers to an invalid *aiocb* structure returns -1 and sets *errno* to [EINVAL].
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

7 **IF** not *PCTS_aio_return* **THEN**
TEST: A call to the *aio_return*() function returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for aio_return: PASS, NO_OPTION

6.7.7 Cancel Asynchronous I/O Request

Function: *aio_cancel*()

6.7.7.1 Synopsis

1
M_GA_stdC_proto_decl(int; aio_cancel; int fildes, struct aiocb * aiocbp; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for aio_cancel: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(aio_cancel; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for aio_cancel: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; aio_cancel; aio.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for aio_cancel: PASS, NO_OPTION

4
M_GA_macro_args (aio_cancel; aio.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for aio_cancel: PASS, NO_OPTION

2204 **6.7.7.2 Description**

2205 **5** **IF** *PCTS_aio_cancel* **THEN**
2206 **IF** *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**
2207 **SETUP:** Queue asynchronous I/O requests for one file descriptor that will not complete
2208 before calling the *aio_cancel*() function.
2209 **TEST:** A call to the *aio_cancel*() function will cancel the asynchronous I/O request
2210 currently outstanding against file descriptor *fildes* 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 aio_cancel: PASS, NO_TEST_SUPPORT, NO_OPTION*

2218 **6** **IF** *PCTS_aio_cancel* **THEN**
2219 **IF** *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**
2220 **SETUP:** Queue multiple asynchronous I/O requests for multiple file descriptors that will
2221 not complete signal notification before calling the *aio_cancel*() function.
2222 **TEST:** A call to the *aio_cancel*() function where the *aiocbp* argument is **NULL** cancels
2223 the outstanding cancellable asynchronous I/O requests against the file descriptor
2224 *fildes* 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 aio_cancel: PASS, NO_TEST_SUPPORT, NO_OPTION*

2229 **7** **IF** *PCTS_aio_cancel* **THEN**
2230 **IF** *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**
2231 **TEST:** Any asynchronous I/O requests that cannot be canceled by a call to the
2232 *aio_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 aio_cancel: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION*

2238 **8** **IF** *PCTS_aio_cancel* **THEN**
2239 **IF** *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**
2240 **TEST:** After a call to the *aio_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 aio_cancel: PASS, NO_TEST_SUPPORT, NO_OPTION*

2247 **9** **IF** *PCTS_aio_cancel* **THEN**
2248 **IF** *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**
2249 **TEST:** After a call to the *aio_cancel*() function, for requested operations that are not
2250 successfully canceled, the *aiocbp* is not modified by *aio_cancel*().
2251 **ELSE** *NO_TEST_SUPPORT*
2252 **ELSE** *NO_OPTION*
2253 *Conformance for aio_cancel: PASS, NO_TEST_SUPPORT, NO_OPTION*

2254 **D_1** **IF** *PCTS_aio_cancel* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the results of calling *aio_cancel*() where the *aiocbp* is not **NULL** and the *fildes* argument does not have the same value as the file descriptor with which the asynchronous operation was initiated does so in §6.7.7.2.
ELSE *NO_OPTION*
Conformance for aio_cancel: PASS, NO_OPTION

D_2 IF *PCTS_aio_cancel* **THEN**
TEST: The PCD.1b documents which asynchronous i/o operations are cancellable by calling the *aio_cancel*() function in §6.7.7.2.
ELSE *NO_OPTION*
Conformance for aio_cancel: PASS, NO_OPTION

6.7.7.3 Returns

R_2 IF *PCTS_aio_cancel* **THEN**
TEST: The *aio_cancel*() function returns the value AIO_CANCELED to the calling process if the requested operation(s) were canceled.
ELSE *NO_OPTION*
SEE: Assertions in §6.7.7.2.

10 IF *PCTS_aio_cancel* **THEN**
IF *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**
TEST: After a call to the *aio_cancel*() function, the value AIO_NOTCANCELED is returned when at least one of the requested operation(s) cannot be canceled because it is in progress.
NOTE: There is no known portable test method for assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_cancel: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

11 IF *PCTS_aio_cancel* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
TEST: The value AIO_ALLDONE is returned for a call to the *aio_cancel*() function when all of the operations have already completed before they could be canceled.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_cancel: PASS, NO_TEST_SUPPORT, NO_OPTION

R_3 IF *PCTS_aio_cancel* **THEN**
TEST: When an error occurs in a call to the *aio_cancel*() function it returns -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: Assertions in §6.7.7.4.

6.7.7.4 Errors

12 IF *PCTS_aio_cancel* **THEN**
IF *PCTS_AIO_CANCELABLE_OPS* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio*) **THEN**
TEST: A call to the *aio_cancel*() function where the *fildes* argument is not a valid file descriptor returns -1 and sets *errno* to [EBADF].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_cancel: PASS, NO_OPTION

13 IF not *PCTS_aio_cancel* **THEN**
TEST: A call to the *aio_cancel*() function returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*

Conformance for aio_cancel: PASS, NO_OPTION

6.7.8 Wait for Asynchronous I/O Request

Function: *aio_suspend*()

6.7.8.1 Synopsis

1

M_GA_stdC_proto_decl(int; aio_suspend; const struct aiocb * const list[], int nent, const struct timespec *timeout; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for aio_suspend: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(aio_suspend; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for aio_suspend: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; aio_suspend; aio.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for aio_suspend: PASS, NO_OPTION

4

M_GA_macro_args (aio_suspend; aio.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for aio_suspend: PASS, NO_OPTION

6.7.8.2 Description

completion

IF *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous I/O operations that will not send signals when they complete and that will not complete until after *aio_suspend*() has been called.
TEST: A call to the *aio_suspend*() function with the *timeout* argument equal to **NULL** suspends the calling process until at least one of the asynchronous I/O operations referenced by the *list* argument has completed and returns zero.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_suspend: PASS, NO_TEST_SUPPORT, NO_OPTION

interrupt

IF *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous I/O operations that will send signals when they complete and that will not complete until after *aio_suspend*() has been called.
TEST: A call to the *aio_suspend*() function with the *timeout* argument equal to **NULL** suspends the calling process until a signal interrupts the function and returns -1 and sets *errno* to [EINTR].
TR: Test for a signal coming from the completion of an asynchronous I/O operation and for a signal coming from another process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for aio_suspend: PASS, NO_TEST_SUPPORT, NO_OPTION

timeout

IF *PCTS_aio_suspend* **THEN**

IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**

SETUP: Queue asynchronous I/O operations that will not complete pass the list of the operations to the *aio_suspend*() function in the test.

TEST: A call to the *aio_suspend*() function with the *timeout* argument not equal to **NULL** suspends the calling process until the time interval specified by *timeout* has passed and returns -1 and sets *errno* to [EAGAIN].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_suspend: PASS, NO_TEST_SUPPORT, NO_OPTION

already_completed

IF *PCTS_aio_suspend* **THEN**

IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**

SETUP: Queue asynchronous I/O operations so that at least one of them will complete before *aio_suspend*() is called.

TEST: A call to the *aio_suspend*() function where any of the *aiocb* structures in the *list* argument correspond to completed asynchronous I/O operations (i.e., the error status for the operation is not equal to [EINPROGRESS]) at the time of the call returns without suspending the calling process and returns zero.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_suspend: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_aio_suspend* **THEN**

IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**

TEST: A call to the *aio_suspend*() function ignores any **NULL** pointers in the *list* argument.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_suspend: PASS, NO_TEST_SUPPORT, NO_OPTION

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

TEST: A PCD.1b that documents the effect of a call to the *aio_suspend*() function where the *list* argument contains pointers that refer to *aiocb* structures that have not been used in submitting asynchronous I/O does so in §6.7.8.2.

ELSE *NO_OPTION*

Conformance for aio_suspend: PASS, NO_OPTION

R_1 **IF** *PCTS_aio_suspend* **THEN**

IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**

TEST: After a call to the *aio_suspend*() function and after the time interval indicated in the *timespec* structure pointed to by *timeout* passes before any of the I/O operations referenced by *list* are completed, the *aio_suspend*() returns with an error.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion timeout in §6.7.8.2.

6.7.8.3 Returns

R_2 **IF** *PCTS_aio_suspend* **THEN**

IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**

TEST: A call to the *aio_suspend*() function returns after one or more asynchronous I/O operations have completed and returns to zero.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertions completion and already_completed in §6.7.8.2.

R_3 IF *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
TEST: A call to the *aio_suspend*() function returns a value of -1 and sets *errno* to indicate the error when an error condition is detected.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertions timeout and interrupt in §6.7.8.2 and no_support in §6.7.8.4.

6.7.8.4 Errors

R_4 IF *PCTS_aio_suspend* **THEN**
IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
TEST: A call to the *aio_suspend*() where no asynchronous I/O indicated in the list referenced by *list* completed in the time interval indicated by *timeout* returns -1 and sets *errno* to [EAGAIN].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion timeout in §6.7.8.2.

6 **TEST:** A call to *aio_suspend*() where a signal interrupts the *aio_suspend*() function returns -1 and sets *errno* to [EINTR].
SEE: Assertion interrupt in §6.7.8.2.
Conformance for aio_suspend: *PASS*

no_support
IF not *PCTS_aio_suspend* THEN
TEST: A call to the *aio_suspend*() function returns -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for aio_suspend: PASS, NO_OPTION

6.7.9 Asynchronous File Synchronization

Function: *aio_fsync*()

6.7.9.1 Synopsis

1
M_GA_stdC_proto_decl(int; aio_fsync; int op, struct aiocb * aiocbp; aio.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for aio_fsync: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(aio_fsync; aio.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for aio_fsync: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; aio_fsync; aio.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for aio_fsync: PASS, NO_OPTION

4
M_GA_macro_args (aio_fsync; aio.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for aio_fsync: PASS, NO_OPTION

6.7.9.2 Description

5 **IF** *PCTS_aio_fsync* **THEN**
IF *PCTS_aio_read* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous read operations using as many of *PCTS_aio_read* and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O data synchronization has not been set.
TEST: A call to the *aio_fsync*() function with the *op* argument equal to O_DSYNC asynchronously forces all read operations associated with the file indicated by the file descriptor *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*() to the synchronized I/O data completion state and returns zero when the synchronization request has been initiated or queued to the file or device. That is, the read operation either completes by transferring an image of the data to the requesting process or, if unsuccessful, by diagnosing and returning an indicator of the error.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityRead in §2.2.2.119
Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** *PCTS_aio_fsync* **THEN**
IF *PCTS_aio_read* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous read operations using as many of *PCTS_aio_read* and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O data synchronization has not been set.
TEST: A call to the *aio_fsync*() function with the *op* argument equal to O_DSYNC asynchronously forces all read operations associated with the file indicated by the file descriptor *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*() to the synchronized I/O data completion state and returns zero when the synchronization request has been initiated or queued to the file or device. That is, at the time that the synchronized read operation initiated by calling *aio_fsync*() any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWbeforeR in §2.2.2.119
Conformance for aio_fsync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_aio_fsync* **THEN**
IF *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous write operations using as many of *PCTS_aio_write* and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O data synchronization has not been set.
TEST: A call to the *aio_fsync*() function with the *op* argument equal to O_DSYNC asynchronously forces all write operations associated with the file indicated by the file descriptor *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*() to the synchronized I/O data completion state and returns zero when the synchronization request has been initiated or queued to the file or device. That is, a write operation initiated by calling *aio_fsync*() either completes by transferring an image of the data to the physical medium containing the file or, if unsuccessful, by diagnosing and returning and indicator of the error.
TR: Test for regular files and, if PCTS_GTI_DEVICE, terminals.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWrite in §2.2.2.119

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_aio_fsync* **THEN**
IF *PCTS_aio_read* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous read operations using as many of *PCTS_aio_read* and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O data synchronization has not been set.
TEST: A call to the *aio_fsync*() function with the *op* argument equal to O_SYNC asynchronously forces all read operations associated with the file indicated by the file descriptor *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*() to the synchronized I/O data completion state and returns zero when the synchronization request has been initiated or queued to the file or device. That is: At the time that the synchronized read operation initiated by calling *aio_fsync*() occurs, any pending write requests affecting the data to be read are written to the physical medium containing the file prior to reading the data and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

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

NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIOFileIntegrityRead in §2.2.2.120
Conformance for aio_fsync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_aio_fsync* **THEN**
IF *PCTS_aio_write* or *PCTS_lio_listio* **THEN**
SETUP: Queue asynchronous write operations using as many of *PCTS_aio_write* and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O data synchronization has not been set.
TEST: A call to the *aio_fsync*() function with the *op* argument equal to O_SYNC asynchronously forces all write operations associated with the file indicated by the file descriptor *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*() to the synchronized I/O file completion state and returns zero when the synchronization request has been initiated or queued to the file or device. That is: At the time that the synchronized write operation initiated by calling *aio_fsync*() occurs, the data are written to the physical medium containing the file and the following file attributes are also written to the physical medium containing the file prior to returning to the calling process:

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

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion GA_syncIOFileIntegrityWrite in §2.2.2.120

Conformance for aio_fsync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

10 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_aio_read* or *PCTS_write* or *PCTS_lio_listio* **THEN**

TEST: After a call to the *aio_fsync*() function and after the request is queued, the error status for all asynchronous I/O operations associated with the file indicated by the file descriptor *aio-fildes* member of the *aiocb* structure referenced by the *aiocbp* argument and queued at the time of the call will be [EINPROGRESS], and the *aio-fsynch*() call returns zero.

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_aio_read* or *PCTS_write* or *PCTS_lio_listio* **THEN**

TEST: After a call to the *aio_fsync*() function and after all data has been successfully transferred, the error status of each queued I/O operation is reset to reflect the success or failure of the operation.

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation. Test for operations that succeed and that fail.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

12 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_aio_read* or *PCTS_write* or *PCTS_lio_listio* **THEN**

SETUP: Queue asynchronous operations, including some write operations, using as many of *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O synchronization has not been set.

TEST: After a call to the *aio_fsync*() function where the *aio-sigevent.sigev_signo* is nonzero, a signal is generated when all operations have achieved synchronized I/O completion, and the *aio_fsync*() call returns zero.

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNCH.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_aio_read* or *PCTS_write* or *PCTS_lio_listio* **THEN**

SETUP: Queue asynchronous operations, including some write operations using as many of *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation against a file for which I/O synchronized has not been set.

TEST: During a call to the *aio_fsync*() function all members of the structure referenced by *aiocbp* are ignored except for *aio_fildes* and *aio_sigevent.*

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNC.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

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

TEST: A PCD.1b that documents the behavior of an implementation when the control block referenced by *aiocbp* in a call to the *aio_fsync*() function becomes an illegal address prior to asynchronous I/O completion does so in §6.7.9.2.

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_OPTION

NOTE: The following statement in POSIX.1b {3} does not make sense since there is no file to synchronize:

If *aiocbp* is **NULL**, then no status is returned in *aiocbp*, and no signal is generated upon completion of the operation.

6.7.9.3 Returns

R_1 **IF** *PCTS_aio_fsync* **THEN**

TEST: The *aio_fsync*() function returns the value 0 to the calling process when the I/O operation is successfully queued.

ELSE *NO_OPTION*

SEE: Assertions in §6.7.9.2.

R_2 **IF** *PCTS_aio_fsync* **THEN**

TEST: The *aio_fsync*() function returns the value -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: Assertions in §6.7.9.4.

6.7.9.4 Errors

14 **IF** *PCTS_aio_fsync* **THEN**

IF *{AIO_MAX}*≤ *PCTS_AIO_MAX* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio)* **THEN**

SETUP: Queue {AIO_MAX} asynchronous I/O operations that will not complete for a file that does not have its synchronized I/O attributes set.

TEST: A call to the *aio_fsync*() function when the requested asynchronous operation was not queued due to temporary resource limitations returns -1 and sets *errno* to [EAGAIN].

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNC.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_aio_fsync* **THEN**

IF *{AIO_MAX}* ≤ *PCTS_AIO_MAX* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio)* **THEN**

SETUP: Queue *PCTS_AIO_MAX* asynchronous I/O operations that will not complete for a file that does not have its synchronized I/O attributes set.

TEST: A call to the *aio_fsync*() returns zero and does not set *errno* to [EAGAIN].

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNC.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio* **THEN**

TEST: A call to the *aio_fsync*() where the *aio_fildes* member of the *aiocb* structure referenced by the *aiocbp* argument is not a valid file descriptor open for writing returns -1 and sets *errno* to [EBADF].

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNC.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

17 **IF** *PCTS_aio_fsync* **THEN**

IF *PCTS_NO_SYNCH_IO_FILE* and (*PCTS_aio_read* or *PCTS_aio_write* or *PCTS_lio_listio)* **THEN**

SETUP: Queue asynchronous I/O operations for a file for which the implementation does not support synchronized I/O.

TEST: A call to the *aio_fsync*() function for a file for which the implementation does not support synchronized I/O returns -1 and sets *errno* to [EINVAL].

TR: Test for as many of the *PCTS_aio_read*, *PCTS_aio_write*, and *PCTS_lio_listio* as are supported by the implementation. Test for the *op* argument being O_DSYNC and O_SYNC.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_aio_fsync* **THEN**

TEST: A call to the *aio_fsync*() function with a value of *op* other than O_DSYNC or O_SYNC returns -1 and sets *errno* to [EINVAL].

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_OPTION

19 **IF** not *PCTS_aio_fsync* **THEN**

TEST: A call to the *aio_fsync*() function returns -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for aio_fsync: PASS, NO_OPTION

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

There are no POSIX.1b {3} assertions in Section 8 except for subclause 8.2.2.2.

8.2.2 Open a Stream on a File Descriptor

Function: *fdopen*()

8.2.2.1 Synopsis

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

8.2.2.2 Description

D_1 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of the *fdopen*() function when *fildes* refers to a shared memory object does so in §8.2.2.3.
ELSE *NO_OPTION*
Conformance for fdopen: PASS, NO_OPTION

8.2.2.3 Returns

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

8.2.2.4 Errors

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

11.1 Semaphore Characteristics

1 **SETUP:** Include <semaphore.h>.
TEST: The type *sem_t* is defined.
Conformance for sem.hdr: PASS

M_GA_semOpenMaxFD(PCTS_SEM_FILE_DESCRIPTORS; function; PCTS_function) =
IF ***PCTS_SEM_FILE_DESCRIPTORS*** **THEN**
IF ***({OPEN_MAX}***<= *PCTS_OPEN_MAX*) and *PCTS_function* **THEN**
TEST: A process calling *function*() can simultaneously open a combination of files and semaphores totaling at least {OPEN_MAX}.
TR: Test for opening {OPEN_MAX} semaphores.

Test for opening a semaphore after opening {OPEN_MAX} -1 files.

Test for opening a file after opening {OPEN_MAX} -1 semaphores.
ELSE *NO_TEST_SUPPORT*
ELSE NO_OPTION

GA_semOpenMaxFD
FOR: *sem_init*() and *sem_open*()
M_GA_semOpenMaxFD(PCTS_SEM_FILE_DESCRIPTORS; function; PCTS_function) =
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
Conformance for sem_hdr: PASS, NO_TEST_SUPPORT, NO_OPTION

M_GA_semPCTSOpenMaxFD(PCTS_SEM_FILE_DESCRIPTORS; function; PCTS_function) =
IF *PCTS_SEM_FILE_DESCRIPTORS* **THEN**
IF ({OPEN_MAX}<= *PCTS_OPEN_MAX*) and *PCTS_function* **THEN**
TEST: A process calling *function*() can simultaneously open a combination of files and semaphores totaling at least *PCTS_OPEN_MAX*.
TR: Test for opening *PCTS_OPEN_MAX*-1 files.

Test for opening a file, after opening {OPEN_MAX} -1 files.

Test for opening a file after opening *PCTS_OPEN_MAX* -1 semaphores.
ELSE *NO_TEST_SUPPORT*
ELSE NO_OPTION

GAsemPCTSOpenMaxFD
FOR: *sem_init*() and *sem_open*()
M_GA_semPCTSOpenMaxFD(PCTS_SEM_FILE_DESCRIPTORS; function; PCTS_function)
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause.

The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

Conformance for sem_hdr: PASS, NO_TEST_SUPPORT, NO_OPTION

11.2 Semaphore Functions

11.2.1 Initialize an Unnamed Semaphore

Function: *sem_init*()

11.2.1.1 Synopsis

1

M_GA_stdC_proto_decl(int; sem_init; sem_t *sem, int pshared, unsigned int value; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sem_init: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sem_init; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sem_init: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sem_init; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sem_init: PASS, NO_OPTION

4

M_GA_macro_args (sem_init; semaphore.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sem_init: PASS, NO_OPTION

11.2.1.2 Description

M_GA_semOpenMaxFD(PCTS_SEM_INIT_FD; sem_init; PCTS_sem_init)
SEE: Assertion GA_semOpenMaxFD in §11.1
Conformance for sem_init: PASS/OpenMaxSems, PCTSOpenMaxSems]

M_GA_semPCTSOpenMaxFD(PCTS_SEM_INIT_FD; sem_init; PCTS_sem_init)
SEE: Assertion GA_semPCTSOpenMaxFD in §11.1
Conformance for sem_init: PASS/OpenMaxSems, PCTSOpenMaxSems]

sem_init

IF *PCTS_sem_init* **THEN**
IF: *PCTS_GAP_sem_init* **THEN**
TEST: A call *sim_init* (*sem, pshared, val)* initializes the unnamed semaphore *sem* to the value of *val.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **FOR: *sem_wait*(), *sem_trywait*(), *sem_post*(), *and sem_destroy* ()**
IF *PCTS_sem_init* **THEN**
IF: *PCTS_GAP_sem_init* and *PCTS_function* **THEN**
TEST: The interface *function*() returns successfully when using a semaphore created by *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function

specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** *PCTS_sem_init* **THEN**

IF: *PCTS_GAP_sem_init* **THEN**

TEST: A semaphore created by *sem_init*() can be used until it is destroyed.

TR: Create a semaphore with a positive value, decrement to zero, then increment and re-decrement to zero.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

7 **FOR:** *sem_wait*(), *sem_trywait*(), *sem_post*(), and *sem_destroy*()

IF *PCTS_sem_init* **THEN**

IF: *PCTS_GAP_sem_init* and *PCTS_function* **THEN**

SETUP: Create a semaphore by calling *sem_init*(*sem, pshared, val*), with a non-zero value of *pshared.*

TEST: Any process that can access the semaphore *sem* can use it for performing *function*().

TR: Test with values {INT_MAX} and {INT_MIN} for *pshared.*

Perform *function*() on the semaphore from the same process that created it and from a child process for each of these values of *pshared.*

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

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

TEST: A PCD.1b that documents the result of referring to copies of *sem* in calls to *sem_wait*(), *sem_trywait*(), *sem_post*(), and *sem_destroy*() does so in §11.2.1.2.

ELSE *NO_OPTION*

Conformance for sem_init: PASS, NO_OPTION

D_2 **IF** *PCTS_sem_init* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the result of a *pshared* argument of zero does so in §11.2.1.2.

ELSE *NO_OPTION*

Conformance for sem_init: PASS, NO_OPTION

D_3 **IF** *PCTS_sem_init* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *sem_init*() function does so in §11.2.1.2.

ELSE *NO_OPTION*

Conformance for sem_init: PASS, NO_OPTION

11.2.1.3 Returns

R_1 **IF** *PCTS_sem_init* **THEN**

IF *PCTS_GAP_sem_init* **THEN**

TEST: When a call to *sem_init*() completes successfully, the semaphore *sem* is initialized.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*
SEE: Assertion sem_init in §11.2.1.2

R_2 IF *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* **THEN**
TEST: When a call to *sem_init*() completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and does not initialize the semaphore.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: All assertions in §11.2.1.4

11.2.1.4 Errors

8 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* and {SEM_VALUE_MAX}< {UINT_MAX} **THEN**
TEST: A call to *sem_init(sem, pshared, val)* when *val* exceeds {SEM_VALUE_MAX} returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* **THEN**
TEST: When a resource required to initialize the semaphore has been exhausted, a call to *sem_init*() returns a value of -1 and sets *errno* to [ENOSPC].
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

10 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* and *PCTS_SEM_IS_FD* and ({OPEN_MAX} < *PCTS_OPEN_MAX)* **THEN**
SETUP: Open {OPEN_MAX} files. Try to open a semaphore.
TEST: When a resource required to initialize the semaphore has been exhausted, a call to *sem_init*() returns a value of -1 and sets *errno* to [ENOSPC].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* and *PCTS_SEM_IS_FD* and ({OPEN_MAX} >= *PCTS_OPEN_MAX)* **THEN**
SETUP: Open *PCTS_OPEN_MAX* files. Try to open a semaphore.
TEST: When a resource required to initialize the semaphore has been exhausted, a call to *sem_init*() returns a value of -1 and sets *errno* to [ENOSPC].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

12 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* and {SEN_NSEMS_MAX}< *PCTS_SEM_NSEMS_MAX* **THEN**
TEST: A call to *sem_init*(), after {SEM_NSEMS_MAX} semaphores have been created, returns a value of -1 and sets *errno* to [ENOSPC].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* and {SEN_NSEMS_MAX}>= *PCTS_SEM_NSEMS_MAX* **THEN**
TEST: A call to *sem_init*(), after {PCTS_SEM_NSEMS_MAX} semaphores have been created, returns a value of -1 and sets *errno* to [ENOSPC].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

14 **IF** not *PCTS_sem_init* **THEN**
IF *PCTS_GAP_sem_init* **THEN**
TEST: A call to *sem_init*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_sem_init* **THEN**
IF *PCTS_RAP_sem_init* **THEN**
TEST: A call to *sem_init*(), when the process lacks the appropriate privileges to initialize a semaphore, returns a value of -1 and sets *errno* to [EPERM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_init: PASS, NO_TEST_SUPPORT, NO_OPTION

11.2.2 Destroy an Unnamed Semaphore

Function: *sem_destroy*()

11.2.2.1 Synopsis

1
M_GA_stdC_proto_decl(int; sem_destroy; sem_t *sem, semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sem_destroy: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(sem_destroy; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sem_destroy: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; sem_destroy; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sem_destroy: PASS, NO_OPTION

4
M_GA_macro_args (sem_destroy; semaphore.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sem_destroy: PASS, NO_OPTION

11.2.2.2 Description

sem_destroy
IF *PCTS_sem_destroy* **THEN**
TEST: A call *sem_destroy(sem)* destroys the unnamed semaphore *sem* and returns 0.
ELSE *NO_OPTION*
Conformance for sem_destroy: PASS, NO_OPTION

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

TEST: A PCD.1b that documents the effect of calling *sem_destroy*() with a named semaphore does so in §11.2.2.2.

ELSE *NO_OPTION*

Conformance for sem_destroy: PASS, NO_OPTION

D_2 IF *PCTS_sem_destroy* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the effect of using the semaphore sem after it is destroyed by a call to *sem_destroy()* does so in §11.2.2.2.

ELSE *NO_OPTION*

Conformance for sem_destroy: PASS, NO_OPTION

D_3 IF *PCTS_sem_destroy* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the effect of destroying a semaphore upon which other processes are currently blocked does so in §11.2.2.2.

ELSE *NO_OPTION*

Conformance for sem_destroy: PASS, NO_OPTION

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

TEST: A PCD.1b that documents whether or not it supports the *sem_destroy*() function does so in §11.2.2.2.

ELSE *NO_OPTION*

Conformance for sem_destroy: PASS, NO_OPTION

11.2.2.3 Returns

R_1 IF *PCTS_sem_destroy* **THEN**

TEST: When a call to *sem_destroy*() completes successfully, the interface returns a value of 0.

ELSE *NO_OPTION*

SEE: Assertion sem_destroy in §11.2.2.2

R_2 IF *PCTS_sem_destroy* **THEN**

TEST: When a call to *sem_destroy*() completes unsuccessfully, interface returns a value of -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: All assertions in §11.2.2.4

11.2.2.4 Errors

5 **IF** *PCTS_sem_destroy* **THEN**

TEST: The call *sem_destroy(sem),* when the argument *sem* is not a valid semaphore, returns a value of -1 and sets *errno* to [EINVAL].

NOTE: A subroutine is recommended that either returns an invalid semaphore or indicates that there is no way to generate an invalid semaphore on the system.

ELSE *NO_OPTION*

Conformance for sem_destroy: PASS, NO_OPTION

6 **IF** not *PCTS_sem_destroy* **THEN**

TEST: A call to *sem_destroy*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for sem_destroy: PASS, NO_OPTION

7 **IF** *PCTS_sem_destroy* and *PCTS_SEM_EBUSY* **THEN**

TEST: A call to *sem_destroy(sem),* when there are currently processes blocked on the semaphore, returns a value of -1 and sets *errno* to [EBUSY].

ELSE *NO_OPTION*

Conformance for sem_destroy: PASS, NO_OPTION

11.2.3 Initialize/Open a Named Semaphore

Function: *sem_open*()

11.2.3.1 Synopsis

1

M_GA_stdC_proto_decl(sem_t *; sem_open; const char *name, int oflag, ...; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sem_open: PASS[1, 2], NO_OPTION

2

M_GA_commonC_result_decl(sem_t*; sem_open; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sem_open: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(sem_t*; sem_open; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sem_open: PASS, NO_OPTION

4

M_GA_macro_args (sem_open; semaphore.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sem_open: PASS, NO_OPTION

11.2.3.2 Description

M_GA_semOpenMaxFD(PCTS_SEM_OPEN_FD; sem_open; PCTS_sem_open)
SEE: Assertion GA_semOpenMaxFD in §11.1
Conformance for sem_open: PASS/OpenMaxSems, PCTSOpenMaxSems]

M_GA_semPCTSOpenMaxFD(PCTS_SEM_OPEN_FD; sem_open; PCTS_sem_open)
SEE: Assertion GA_semPCTSOpenMaxFD in §11.1
Conformance for sem_open: PASS/OpenMaxSems, PCTSOpenMaxSems]

sem_open
FOR: *sem_wait*(), *sem_trywait*(), *sem_post*(), and *sem_close*()
IF *PCTS_sem_open* **THEN**
IF *PCTS_function* **THEN**
TEST: A call *sem_open(name, oflag, ...)* establishes a connection between a named semaphore, *name*, and the calling process and returns the address of the named semaphore that can be used in subsequent calls to *function*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_sem_open* **THEN**
SETUP: Include the header `<semaphore.h>`.
TEST: The constants O_CREAT and O_EXCL are defined and are bitwise distinct.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

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

IF *PCTS_sem_open* **THEN**
SETUP: Create a semaphore using *sem_open*().
TEST: The semaphore remains usable by the calling process until it is closed by a successful call to *function*().
TR: Test using *sem_wait*(), *sem_trywait*(), and *sem_post*()
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

7 **IF** *PCTS_sem_open* **THEN**
SETUP: Open a semaphore by calling *sem_open(name, oflag, ...)* where the flag bit O_CREAT is set in *oflag* and the named semaphore *name* does not already exist.
TEST: The named semaphore is created.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

8 **IF** *PCTS_sem_open* **THEN**
SETUP: The call *sem_open(name, oflag, mode, value)* creates a semaphore with an initial value of *value.*
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

9 **IF** *PCTS_sem_open* **THEN**
TEST: Valid initial values for semaphores are unsigned integers with values between 0 and {SEM_VALUE_MAX}, inclusive.
TR: Test values of 0, {SEM_VALUE_MAX}, and {SEM_VALUE_MAX}+1, if it is less than {UINT_MAX}.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

10 **IF** *PCTS_sem_open* **THEN**
TEST: The user ID of the semaphore created by *sem_open*() is the effective user ID of the process.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

11 **IF** *PCTS_sem_open* **THEN**
TEST: The group ID of the semaphore created by *sem_open*() is a system default group ID or the effective group ID of the process.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

12 **IF** *PCTS_sem_open* **THEN**
TEST: The permission bits of the semaphore created by *sem_open(name, oflag, mode, value)* are set to the value of the *mode* argument, except those set in the file mode creation mask of the process.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_1 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect of calling *sem_open (name, oflag, mode, value)* with bits specified in *mode* other than the file permission bits does so in §11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

13 **IF** *PCTS_sem_open* **THEN**

SETUP: Create a new named semaphore with the call *sem_open(name, oflag, ...)* and the O_CREAT flag set in *oflag.*
TEST: Other processes can connect to the semaphore by calling *sem_open*() with the same value of *name.*
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

14 **IF** *PCTS_sem_open* **THEN**
TEST: When the semaphore *name* exists, the call *sem_open (name, oflag, ...)* with the O_CREAT and O_EXCL flags set in *oflag,* fails.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

15 **IF** *PCTS_sem_open* **THEN**
TEST: The check for the existence of the semaphore in a call to *sem_open( )* with o_excl and O_CREAT flag set, and the creation of the semaphore if it does not exist, are atomic with respect to other processes executing *sem_open*() with O_EXCL and O_CREAT set.
NOTE: There is no known reliable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_TEST, NO_OPTION

D_2 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect of calling *sem_open*() with O_EXCL set and O_CREAT not set does so in §11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_3 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect of calling *sem_open*() with oflag other than O_EXCL and O_CREAT specified in the *oflag* parameter, does so in §11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_4 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether *name* appears in the filesystem does so in §11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_5 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether *name* is visible to functions that take pathnames as arguments does so in §11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

16 ***M_GA_portableFilenames(sem_open)***
SEE: Assertion GA_portableFilenames in §2.2.2.40
Conformance for sem_open: PASS, NO_OPTION

17 ***M_GA_upperLowerNames(sem_open)***
SEE: Assertion GA_upperLowerNames in §2.2.2.40
Conformance for sem_open: PASS, NO_OPTION

18 ***M_GA_PRNoTrunc(sem_open)***
SEE: Assertion GA_PRNoTrunc in §2.3.6
Conformance for sem_open: PASS, NO_OPTION

19 ***M_GA_PRNoTruncError(sem_open)***
SEE: Assertion GA_PRNoTruncError in §2.3.6

Conformance for sem_open: PASS, NO_OPTION

20 **IF** *PCTS_sem_open* **THEN**
SETUP: Call *sem_open(name, oflag, ...)* where *name* begins with the slash character.
TEST: Processes calling *sem_open*() with the same value of *name* refer to the same semaphore object, as long as that name has not been removed.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_6 **IF** *PCTS_sem_open* **THEN**
TEST: A PCD.1b documents the effect of calling *sem_open(name, oflag, …)* when *name* does not begin with a slash character in §11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_7 **IF** *PCTS_sem_open* **THEN**
TEST: A PCD.1b documents the interpretation of slash characters in §11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

21 **IF** *PCTS_sem_open* **THEN**
SETUP: Call *sem_open(name, oflag, ...)* with a single value for *name* and no intervening calls to *sem_unlink*().
TEST: The same semaphore address is returned for each successful call.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_8 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effects of references to copies of semaphores created with *sem_open*() does so in §11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

D_9 **IF** *PCTS_sem_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that whether or not it supports the *sem_open*() function does so in §11.2.3.2.
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

11.2.3.3 Returns

R_1 **IF** *PCTS_sem_open* **THEN**
TEST: When a call to *sem_open* () completes successfully, the address of the semaphore is returned.
ELSE *NO_OPTION*
SEE: Assertion sem_open in §11.2.3.2

11.2.3.4 Errors

22 **IF** *PCTS_sem_open* **THEN**
TEST: A call to *sem_open*() when the named semaphore exists and the permissions specified by *oflag* are denied, or the named semaphore does not exist and permission to create the named semaphore is denied, returns a value of -1 and sets *errno* to [EACCESS]
ELSE *NO_OPTION*
Conformance for sem_open: PASS, NO_OPTION

23 **IF** *PCTS_sem_open* **THEN**

TEST: A call to *sem_open*() when O_CREAT and O_EXCL are set and the named semaphore already exists, returns a value of -1 and sets *errno* to [EEXIST]

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

24 **IF** *PCTS_sem_open* **THEN**

TEST: A call to *sem_open*() when the *sem_open*() operation is interrupted by a signal returns a value of -1 and sets *errno* to [EINTR]

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

25 **IF** *PCTS_sem_open* **THEN**

TEST: A call to *sem_open*() when the *sem_open*() operation is not supported for the given name returns a value of -1 and sets *errno* to [EINVAL]

NOTE: A subroutine is recommended that either returns a name for which *sem_open*() is not supported or indicates that there is no way to generate *sem_open*() on the system.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

D_10 **IF** *PCTS_sem_open* **THEN**

TEST: The PCD.1b documents under what circumstances [EINVAL] may be returned in §11.2.3.4.

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

26 **IF** *PCTS_sem_open* **THEN**

TEST: A call to *sem_open(name, oflag, ...)* when O_CREAT is specified in *oflag* and *value* is greater than {SEM_VALUE_MAX} returns a value of -1 and sets *errno* to [EINVAL]

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

27 **IF** *PCTS_sem_open* **THEN**

TEST: A call to *sem_open*() when too many semaphore descriptors are currently in use by this process returns a value of -1 and sets *errno* to [EMFILE]

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

28 **IF** *PCTS_sem_open* **THEN**

IF *PCTS_SEM_IS_FD* and ({OPEN_MAX}< *PCTS_OPEN_MAX)* **THEN**

TEST: A call to *sem_open*() when too many file descriptors are currently in use by this process returns a value of -1 and sets *errno* to [EMFILE]

TR: Open {OPEN_MAX} files. Try to open a semaphore.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_TEST_SUPPORT, NO_OPTION

29 **IF** *PCTS_sem_open* **THEN**

IF {PATH_MAX}<= *PCTS_PATH_MAX)* **THEN**

TEST: A call to *sem_open*() when the length of the *name* string exceeds {PATH_MAX} returns a value of -1 and sets *errno* to [ENAMETOOLONG]

TR: Open {OPEN_MAX} files. Try to open a semaphore.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_TEST_SUPPORT, NO_OPTION

30 **IF** *PCTS_sem_open* **THEN**

IF {NAME_MAX}<= *PCTS_NAME_MAX)* and {_POSIX_NO_TRUNC} **THEN**

TEST: A call to *sem_open*() when a pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC) is in effect returns a value of -1 and sets *errno* to [ENAMETOOLONG]

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_TEST_SUPPORT, NO_OPTION

31 **IF** *PCTS_sem_open* **THEN**

TEST: A call to *sem_open*() when too many semaphores are currently open in the system returns a value of -1 and sets *errno* to [ENFILE]

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

32 **IF** *PCTS_sem_open* **THEN**

TEST: A call to *sem_open*() when O_CREAT is not set and the named semaphore does not exist returns a value of -1 and sets *errno* to [ENOENT]

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

33 **IF** *PCTS_sem_open* **THEN**

TEST: A call to *sem_open*() when there is insufficient space for the creation of the new named semaphore returns a value of -1 and sets *errno* to [ENOSPC]

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

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_TEST, NO_OPTION

34 **IF** not *PCTS_sem_open* **THEN**

TEST: A call to *sem_open*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for sem_open: PASS, NO_OPTION

11.2.4 Close a Named Semaphore

Function: *sem_close*()

11.2.4.1 Synopsis

1

M_GA_stdC_proto_decl(int; sem_close; sem_t *sem; semaphore.h;;;)

SEE: Assertion GA_stdC_proto_decl in §2.7.3

Conformance for sem_close: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sem_close; semaphore.h;;;)

SEE: Assertion GA_commonC_int_result_decl in §2.7.3

Conformance for sem_close: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sem_close; semaphore.h;;;)

SEE: Assertion GA_macro_result_decl in §1.3.4

Conformance for sem_close: PASS, NO_OPTION

4

M_GA_macro_args (sem_close; semaphore.h;;;)

SEE: Assertion GA_macro_args in §2.7.3

Conformance for sem_close: PASS, NO_OPTION

11.2.4.2 Description

sem_close

IF *PCTS_sem_close* **THEN**
IF: *PCTS_sem_open* and {SEN_NSEMS_MAX} < *PCTS_SEM_NSEMS_MAX* **THEN**
TEST: The *sem_close*() function makes any system resources allocated by the system available for reuse by a subsequent *sem_open*() call in this process and returns zero.
TR: Open {SEM_NSEMS_MAX} semaphores, close one, then open another.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 IF *PCTS_sem_close* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effects of calling *sem_close*() for an unnamed semaphore does so in §11.2.4.2.
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

D_2 IF *PCTS_sem_close* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effects subsequent use of the semaphore *sem* does so in §11.2.4.2.
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

5 **IF** *PCTS_sem_close* **THEN**
TEST: The *sem_close*() has no effect on the state of the semaphore, if the semaphore has not been removed with a successful call to *sem_unlink*().
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

6 **IF** *PCTS_sem_close* **THEN**
IF: *PCTS_sem_open* *PCTS_sem_unlink* **THEN**
SETUP: Call *sem_open*() with O_CREAT, then call *sem_unlink*() successfully on the same semaphore.
TEST: When all processes that have opened the semaphore close it, the semaphore is no longer accessible.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_TEST_SUPPORT, NO_OPTION

D_3 IF *PCTS_sem_close* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sem_close*() function does so in §11.2.4.2.
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

11.2.4.3 Returns

R_1 IF *PCTS_sem_close* **THEN**
TEST: When a call to *sem_close*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion sem_close in §11.2.4.2

R_2 IF *PCTS_sem_close* **THEN**
TEST: When a call to *sem_close*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: Assertion sem_close in §11.2.4.4

11.2.4.4 Errors

7 **IF** *PCTS_sem_close* **THEN**
TEST: A call to *sem_close*(), when argument is not a valid semaphore descriptor, returns a value of -1 and sets *errno* to {EINVAL}
NOTE: A subroutine is recommended that either returns an invalid semaphore descriptor or indicates that there is no way to generate an invalid semaphore descriptor on the system.
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

8 **IF** not *PCTS_sem_close* **THEN**
TEST: A call to *sem_close*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sem_close: PASS, NO_OPTION

11.2.5 Remove a Named Semaphore

Function: *sem_unlink*()

11.2.5.1 Synopsis

1
M_GA_stdC_proto_decl(int; sem_unlink; const char *name; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sem_unlink: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(sem_unlink; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sem_unlink: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; sem_unlink; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sem_unlink: PASS, NO_OPTION

4
M_GA_macro_args (sem_unlink; semaphore.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sem_unlink: PASS, NO_OPTION

11.2.5.2 Description

sem_unlink
IF *PCTS_sem_unlink* **THEN**
TEST: The call *sem_unlink(name)* removes the semaphore named by the string *name*, and returns the value zero.
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

5 **IF** *PCTS_sem_unlink* **THEN**
TEST: When the semaphore named by name is currently referenced by other processes, then *sem_unlink(name)* has no effect on the state of the semaphore.
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

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

IF *PCTS_sem_unlink* **THEN**
TEST: When one or more processes has the semaphore open when *sem_unlink*()is called, destruction of the semaphore is postponed until all references to the semaphore have been destroyed by a call to *function*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

7 **IF** *PCTS_sem_unlink* **THEN**
IF *PCTS_sem_open*
TEST: Calls to *sem_open*() to re-create or re-connect to the semaphore refer to a new semaphore after *sem_unlink*() is called.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_sem_unlink* **THEN**
TEST: When any process references name, the call *sem_unlink(name)* returns immediately.
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

D_1 **IF** *PCTS_sem_unlink* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sem_unlink*() function does so in §11.2.5.2.
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

11.2.5.3 Returns

R_1 **IF** *PCTS_sem_unlink* **THEN**
TEST: When a call to *sem_unlink*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion sem_unlink in §11.2.5.2

R_2 **IF** *PCTS_sem_unlink* **THEN**
TEST: When a call to *sem_unlink*() completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and does not change the semaphore.
ELSE *NO_OPTION*
SEE: All assertions in §11.2.5.4

11.2.5.4 Errors

9 **IF** *PCTS_sem_unlink* **THEN**
TEST: A call to *sem_unlink*(), when permission is denied to unlink the named semaphore, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

10 **IF** *PCTS_sem_unlink* **THEN**
IF *{_posix_no_truck} AND {name_max}<=* ***PCTS_NAME_MAX*** **THEN**
TEST: A call to *sem_unlink*(), when the length of the *name* string exceeds {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect, returns a value of -1 and sets *errno* to [ENAMETOOLONG]
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*

Conformance for sem_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** *PCTS_sem_unlink* **THEN**
TEST: A calls to *sem_unlink*(), when the named semaphore does not exist, returns a value of -1 and sets *errno* to [ENOENT].
NOTE: A subroutine is recommended that either returns the name of a semaphore that does not exist or indicates that there is no way to generate the name of a semaphore that does not exist in the system.
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

12 **IF** not *PCTS_sem_unlink* **THEN**
TEST: A calls to *sem_unlink*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sem_unlink: PASS, NO_OPTION

11.2.6 Lock a Semaphore

Functions: *sem_wait*(), *sem_trywait*()

11.2.6.1 Synopsis

1
M_GA_stdC_proto_decl(int; sem_wait; sem_t *sem; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sem_wait: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(sem_wait; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sem_wait: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; sem_wait; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sem_wait: PASS, NO_OPTION

4
M_GA_macro_args (sem_wait; semaphore.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sem_wait: PASS, NO_OPTION

5
M_GA_stdC_proto_decl(int; sem_trywait; sem_t *sem; semaphore.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sem_trywait: PASS[5, 6], NO_OPTION

6
M_GA_commonC_int_result_decl(sem_trywait; sem_t*sem; semaphore.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sem_trywait: PASS[5, 6], NO_OPTION

7
M_GA_macro_result_decl(int; sem_trywait; semaphore.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sem_trywait: PASS, NO_OPTION

8

M_GA_macro_args (sem_trywait; semaphore.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sem_trywait: PASS, NO_OPTION

11.2.6.2 Description

sem_wait
FOR: *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When the call *sem_wait(sem)* locks the semaphore *sem*, with the semaphore lock operation, it returns the value zero.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

sem_trywait
FOR: *sem_init*() and *sem_open*()
IF *PCTS_sem_trywait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When the call *sem_trywait(sem)* locks the semaphore *sem*, with the semaphore lock operation, it returns the value zero.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When the semaphore value is currently zero, and the call to *sem_wait(sem)* is not interrupted by a signal, when the call returns the semaphore referenced by *sem* is locked.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

10 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_trywait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When the semaphore value is currently positive, and the call to *sem_wait(sem)* is not interrupted by a signal, when the call returns the semaphore referenced by *sem* is locked, and the interface returns the value zero.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_trywait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When the semaphore value is currently zero, the call to *sem_trywait*() does not lock the semaphore referenced by *sem*.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

12 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* and *sem_post*()**THEN**
SETUP: Create a semaphore using *function*().
TEST: After a successful *sem_wait*() call, the semaphore is locked and remains locked until the *sem_post*() function is executed and returns successfully.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_trywait* **THEN**
IF *PCTS_function* and *sem_post*() **THEN**
SETUP: Create a semaphore using *function*().
TEST: After a successful call to *sem_trywait*(), the semaphore is locked and remains locked until the *sem_post* function is executed and returns successfully.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

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

IF *PCTS_sem_wait* **THEN**

IF *PCTS_function* **THEN**

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

TEST: The *sem_wait*() function is interruptible by the delivery of a signal.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

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

TEST: A PCD.1b that documents whether or not it supports the *sem_wait*() function does so in §11.2.6.2.

ELSE *NO_OPTION*

Conformance for sem_wait: PASS, NO_OPTION

D_2 **IF** *PCTS_sem_trywait* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *sem_trywait*() function does so in §11.2.6.2.

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_OPTION

11.2.6.3 Returns

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

IF *PCTS_sem_wait* **THEN**

IF *PCTS_function* **THEN**

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

TEST: The *sem_wait(sem)* function returns 0 if the calling process successfully performed the semaphore lock operation on the semaphore designated by *sem.*

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_trywait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: The *sem_trywait(sem)* function returns 0 if the calling process successfully performed the semaphore lock operation on the semaphore designated by *sem.*
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

R_1 FOR: *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When a call to *sem_wait(sem)* completes successfully, the interface returns a value of 0 and the semaphore designated by *sem* is locked by the semaphore lock operation.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion sem_wait in §11.2.6.2

R_2 FOR: *sem_init*() and *sem_open*()
IF *PCTS_sem_trywait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When a call to *sem_trywait(sem)* completes successfully, the interface returns a value of 0 and the semaphore designated by *sem* is locked by the semaphore lock operation.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion sem_trywait in §11.2.6.2

R_3 FOR: *sem_init*() and *sem_open*()
IF *PCTS_sem_wait* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().

TEST: When a call to *sem_wait(sem)* completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and does not change the state of the semaphore.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: All assertions in §11.2.6.4 controlled by *PCTS_sem_wait*

R_4 FOR: *sem_init*() and *sem_open*()

IF *PCTS_sem_trywait* **THEN**

IF *PCTS_function* **THEN**

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

TEST: When a call to *sem_trywait*() completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and does not change the state of the semaphore.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: All assertions in §11.2.6.4 controlled by *PCTS_sem_trywait*

11.2.6.4 Errors

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

IF *PCTS_sem_wait* **THEN**

IF *PCTS_function* **THEN**

TEST: A call to *sem_wait*(), when the semaphore is already locked, so it cannot be immediately locked by the *sem_trywait*() operation, returns a value of -1 and sets *errno* to [EAGAIN].

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

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

IF *PCTS_sem_trywait* **THEN**

IF *PCTS_function* **THEN**

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

TEST: A call to *sem_trywait*(), when the semaphore is already locked, so it cannot be immediately locked by the *sem_trywait*() operation, returns a value of -1 and sets *errno* to [EAGAIN].

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

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

IF *PCTS_sem_wait* **THEN**

IF *PCTS_function* **THEN**

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

TEST: A call to *sem_wait*(), when the *sem* argument does not refer to a valid semaphore returns a value of -1 and sets *errno* to [EINVAL].

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

NOTE: A subroutine is recommended that either returns an invalid semaphore or indicates that there is no way to generate an invalid semaphore on the system.

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

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

IF *PCTS_sem_trywait* **THEN**

IF *PCTS_function* **THEN**

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

TEST: A call to *sem_trywait(sem),* when the *sem* argument does not refer to a valid semaphore returns a value of -1 and sets *errno* to [EINVAL].

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

NOTE: A subroutine is recommended that either returns an invalid semaphore or indicates that there is no way to generate an invalid semaphore on the system.

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

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

IF *PCTS_sem_wait* **THEN**

IF *PCTS_function* **THEN**

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

TEST: A call to *sem_wait*(), interrupted by a signal, returns a value of -1 and sets *errno* to [EINTR].

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

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

IF *PCTS_sem_trywait* **THEN**

IF *PCTS_function* **THEN**

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

TEST: A call to *sem_trywait*(), when interrupted by a signal, returns a value of -1 and sets *errno* to [EINVAL].

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

23 **IF** not *PCTS_sem_wait* **THEN**

TEST: A call to *sem_wait*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for sem_wait: PASS, NO_OPTION

24 **IF** not *PCTS_sem_trywait* **THEN**

TEST: A call to *sem_trywait*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_OPTION

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

IF *PCTS_sem_wait* **THEN**

IF *PCTS_function* **THEN**

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

TEST: A call to *sem_wait*(), when a deadlock condition is detected, returns a value of -1 and sets *errno* to [EDEADLK].

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_wait: PASS, NO_TEST_SUPPORT, NO_OPTION

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

IF *PCTS_sem_trywait* **THEN**

IF *PCTS_function* **THEN**

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

TEST: A call to *sem_trywait*(), when a deadlock condition is detected, returns a value of -1 and sets *errno* to [EDEADLK].

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_trywait: PASS, NO_TEST_SUPPORT, NO_OPTION

11.2.7 Unlock a Semaphore

Function: *sem_post*()

11.2.7.1 Synopsis

1

M_GA_stdC_proto_decl(int; sem_post; sem_t *sem, semaphore.h;;;)

SEE: Assertion GA_stdC_proto_decl in §2.7.3

Conformance for sem_post: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sem_post; semaphore.h;;;)

SEE: Assertion GA_commonC_int_result_decl in §2.7.3

Conformance for sem_post: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sem_post; semaphore.h;;;)

SEE: Assertion GA_macro_result_decl in §1.3.4

Conformance for sem_post: PASS, NO_OPTION

4

M_GA_macro_args (sem_post; semaphore.h;;;)

SEE: Assertion GA_macro_args in §2.7.3

Conformance for sem_post: PASS, NO_OPTION

11.2.7.2 Description

sem_post

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

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* **THEN**

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

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

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

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

5 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_post* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When the semaphore value resulting from *sem_post*() is positive, then the semaphore value is incremented.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_post* **THEN**
IF *PCTS_sem_wait* **THEN**
SETUP: Create a semaphore using *function*(). Also, create multiple processes and have them block waiting on the semaphore.
TEST: When the value of the semaphore resulting from *sem_post*() is zero, then one of the processes blocked waiting for the semaphore returns successfully from its call to *sem_wait*().
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

R_1 FOR: *sem_init*() and *sem_open*()
IF *PCTS_sem_post* **THEN**
IF *PCTS_function* and {_POSIX_PRIORITY_SCHEDULING} **THEN**
SETUP: Create a semaphore using *function*().
TEST: When the value of the semaphore resulting from *sem_post*() is zero, the process to be unblocked is chosen in a manner appropriate to the scheduling policies and parameters in effect for the blocked processes.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion 8 in §11.2.7.2.

7 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_post* **THEN**
IF *PCTS_function* and {_POSIX_PRIORITY_SCHEDULING}**THEN**
SETUP: Create a semaphore using *function*().
TEST: When the value of the semaphore resulting from *sem_post*() is zero, and the scheduler is SCHED_FIFO or SCHED_RR, the highest priority waiting process is unblocked.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

Test for each of SCHED_FIFO and SCHED_RR.

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

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

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* and {_POSIX_PRIORITY_SCHEDULING}**THEN**

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

TEST: When the value of the semaphore resulting from *sem_post*() is zero, and the scheduler is SCHED_FIFO or SCHED_RR, and there is more than one highest priority process blocked waiting for the semaphore, then the highest priority process that has been waiting the longest is unblocked.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

Test for each of SCHED_FIFO and SCHED_RR.

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

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

TEST: A PCD.1b that documents the choice of a process to unblock if {_POSIX_PRIORITY_SCHEDULING} is defined does so in §11.2.7.2.

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_OPTION

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

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* **THEN**

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

TEST: The *sem_post*() function is reentrant with respect to signals and may be invoked from a signal-catching function.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_sem_post* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *sem_post*() function does so in §11.2.7.2.

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_OPTION

11.2.7.3 Returns

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

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* **THEN**

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

TEST: When a call to *sem_post*() completes successfully, the interface returns a value of 0.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion sem_post in §11.2.7.2

R_3 FOR: *sem_init*() and *sem_open*()

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* **THEN**

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

TEST: When a call to *sem_post*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: All assertions in §11.2.7.4

11.2.7.4 Errors

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

IF *PCTS_sem_post* **THEN**

IF *PCTS_function* **THEN**

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

TEST: A call to *sem_post*(), when the *sem* does not refer to a valid semaphore returns a value of -1 and sets *errno* to [EINVAL].

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

NOTE: A subroutine is recommended that either returns an invalid semaphore or indicates that there is no way to generate an invalid semaphore on the system.

The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function

specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **IF** not *PCTS_sem_post* **THEN**

TEST: A call to *sem_post*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for sem_post: PASS, NO_OPTION

11.2.8 Get the Value of a Semaphore

Function: *sem_getvalue*()

11.2.8.1 Synopsis

1

M_GA_stdC_proto_decl(int; sem_getvalue; sem_t *sem, int *sval; semaphore.h;;;)

SEE: Assertion GA_stdC_proto_decl in §2.7.3

Conformance for sem_getvalue: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sem_getvalue; semaphore.h;;;)

SEE: Assertion GA_commonC_int_result_decl in §2.7.3

Conformance for sem_getvalue: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sem_getvalue; semaphore.h;;;)

SEE: Assertion GA_macro_result_decl in §1.3.4

Conformance for sem_getvalue: PASS, NO_OPTION

4

M_GA_macro_args (sem_getvalue; semaphore.h;;;)

SEE: Assertion GA_macro_args in §2.7.3

Conformance for sem_getvalue: PASS, NO_OPTION

11.2.8.2 Description

sem_getvalue

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

IF *PCTS_sem_getvalue* **THEN**

IF *PCTS_function* **THEN**

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

TEST: A successful call of *sem_getvalue(sem, sval)* updates the location referenced by the *sval* argument to have the value of the semaphore referenced by *sem* at some unspecified time during the call, and returns the value to zero.

TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().

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

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_getvalue: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_getvalue* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: The *sem_getvalue*() function does not affect the state of the semaphore referenced by *sem*.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_getvalue: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_getvalue* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When *sem* is locked, then the value returned by *sem_getvalue*() is either zero or a negative number whose absolute value represents the number of processes waiting for the semaphore at some unspecified time during the call.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sem_getvalue: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_sem_getvalue* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sem_getvalue*() function does so in §11.2.8.2.
ELSE *NO_OPTION*
Conformance for sem_getvalue: PASS, NO_OPTION

11.2.8.3 Returns

R_1 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_getvalue* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When a call to *sem_getvalue*() completes successfully, the interface returns a value of 0.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion sem_getvalue in §11.2.8.2

R_2 FOR: *sem_init*() and *sem_open*()
IF *PCTS_sem_getvalue* **THEN**
IF *PCTS_function* **THEN**
SETUP: Create a semaphore using *function*().
TEST: When a call to *sem_getvalue*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: All assertions in §11.2.8.4

7 **FOR:** *sem_init*() and *sem_open*()
IF *PCTS_sem_getvalue* **THEN**
IF *PCTS_function* and *PCTS_SEM_INVALID* **THEN**
SETUP: Create a semaphore using *function*().
TEST: A call to *sem_getvalue*(), when the *sem* argument does not refer to a valid semaphore, returns a value of -1 and sets *errno* to [EINVAL]
TR: When testing for *sem_init*(), perform the test consistent with the flag *PCTS_GAP_sem_init;* that is, generate a *NO_TEST_SUPPORT* test result code if there is not a way to get appropriate privilege to call *sem_init*().
NOTE: A subroutine is recommended that either returns an invalid semaphore or indicates that there is no way to generate an invalid semaphore on the system.

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

8 **IF** not *PCTS_sem_getvalue* **THEN**
TEST: A call to *sem_getvalue*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sem_getvalue: PASS, NO_OPTION

This page is intentionally blank.

Section 12: Memory Management

1 TEST: The page size, in bytes, is the value of the configurable system variable [PAGESIZE].
Conformance for mem_intro: PASS

D_1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents range lockings and mappings are restricted o page-size boundaries, does so in § 12.
ELSE *NO_OPTION*
Conformance for mem_intro: PASS, NO_OPTION

D_2 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents 1B page size, meaning no restrictions on size or alignment, of range lockings and mappings, does so in § 12.
ELSE *NO_OPTION*
Conformance for mem_intro: PASS, NO_OPTION

D_3 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether locking memory guarantees fixed translation between virtual addresses (as seen by the process) and physical addresses, does so in § 12.
ELSE *NO_OPTION*
Conformance for mem_intro: PASS, NO_OPTION

R_1 **IF** *PCTS_mlockall* or *PCTS_mlock* **THEN**
TEST: Per-process memory locks are not inherited across a *fork*( ).
ELSE *NO_TEST_SUPPORT*
SEE: Assertion 2 in §3.1.1.2

R_2 **IF** *PCTS_mlockall* or *PCTS_mlock* **THEN**
TEST: All memory locks owned by a process are unlocked upon *exec* or process termination.
ELSE *NO_TEST_SUPPORT*
SEE: Assertion mlock in §12.1.2.2

R_3 **IF** *PCTS_munlock* **THEN**
TEST: Unmapping of an address range remove any memory locks established on that address range by this process.
ELSE *NO_TEST_SUPPORT*
SEE: Assertion munlock_remove_maps in §12.2.2.2

2 **IF** *PCTS_mmap* **THEN**

TEST: Once a file is "mapped" into a process address space, the data can be manipulated as memory.

ELSE *NO_TEST_SUPPORT*

Conformance for mem_intro: PASS, NO_TEST_SUPPORT

3 **IF** *PCTS_mmap* **THEN**

TEST: When more than one process maps a file, its contents are shared among them..

ELSE *NO_TEST_SUPPORT*

Conformance for mem_intro: PASS, NO_TEST_SUPPORT

4 **IF** *PCTS_mmap* **THEN**

TEST: When the mappings allow shared write access, then data written into the memory object through the address space of one process appear in the address spaces of all processes that similarly map the same portion of the memory object.

ELSE *NO_TEST_SUPPORT*

Conformance for mem_intro: PASS, NO_TEST_SUPPORT

R_4 **IF** *PCTS_shm_unlink* **THEN**

TEST: *unlink* ( ) of a mapped file or *shm-unlink* ( ) of a shared memory object, while causing the removal of the name, does not unmap any mappings while causing the removal of the name, does not unmap any mappings established for the object. Once the name has been removed, the contents of the memory object are preserved as long as a process has the memory object open or has some area of the memory object mapped.

ELSE *NO_TEST_SUPPORT*

SEE: All assertions in §12.3.2.2

R_5 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**

IF *PCTS__mmap* **THEN**

TEST: References to whole pages within the mapping but beyond the current length of an object result in a SIGBUS signal.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion mmap_ SIGBUS in §12.1.2.2

D_4 **IF** {_POSIX_MEMORY_PROTECTION} and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the result of references to memory within the mapping but beyond the current length of an object, does so in §12.

ELSE *NO_OPTION*

Conformance for mem_intro: PASS, NO_OPTION

5 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**

IF: *PCTS_mmap* **THEN**

SETUP: Create a mapped memory object using *mmap* ( ).

TEST: The size of a memory object is unaffected by access beyond the end of the object.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mem_intro: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**

IF: *PCTS_mmap* **THEN**

SETUP: Create a mapped memory object using without write access using *mmap* ( ).

TEST: Write attempts to memory that was mapped without write access, results in a SIGSEGV signal.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mem_intro: PASS, NO_TEST_SUPPORT, NO_OPTION

7 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**

IF: *PCTS_mmap* **THEN**

SETUP: Create a mapped memory object using with PROT_NONE using *mmap* ( ).

TEST: Any access to memory mapped PROT_NONE , results in a SIGSEGV signal.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mem_intro: PASS, NO_TEST_SUPPORT, NO_OPTION

R_6 **IF** {_POSIX_MEMORY_PROTECTION} **THEN**

TEST: References to unmapped addresses result in a SIGSEGV signal.

ELSE *NO_OPTION*

SEE: Assertion munmap_ SIGSEV in §12.2.2.2

D_5 **IF** not {_POSIX_MEMORY_PROTECTION} and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the effect of references to unmapped addresses, does so in §12.

ELSE *NO_OPTION*

Conformance for mem_intro: PASS, NO_OPTION

12.1 Memory Locking Functions

12.1.1 Lock/Unlock the Address Space of a Process

Functions: *mlockall*( ), *munlockall* ()

12.1.1.1 Synopsis

1

M_GA_stdC_proto_decl(int; mlockall; int; int flags; sys/mman .h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for mlockall: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(mlockall; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for mlockall: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; mlockall; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for mlockall: PASS, NO_OPTION

4

M_GA_macro_args (mlockall; sys/mman.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for mlockall: PASS, NO_OPTION

5

M_GA_stdC_proto_decl(int; munlockall; sys/mman .h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for munlockall: PASS[5, 6], NO_OPTION

6

M_GA_commonC_int_result_decl(munlockall; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for munlockall: PASS[5, 6], NO_OPTION

7

M_GA_macro_result_decl(int; munlockall; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for munlockall: PASS, NO_OPTION

8

M_GA_macro_args (munlockall; sys/mman.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for munlockall: PASS, NO_OPTION

12.1.1.2 Description

mlockall

FOR: *exel*( ), *execv*( ), *execle*( ), *execve*( ), *execle*( ), and *execve*( )

IF *PCTS_mlockall* **THEN**

IF: *PCTS_GAP_mlockall* **THEN**

TEST: A successful call to the function *mlockall*() returns zero and makes all of the pages mapped by the address space of a process memory resident until unlocked, or until the process exits or executes a successful call to *function*( ).

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

TR: Try both *exec* and *exit*( ).

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sem_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_mlockall* **THEN**

SETUP: Include the header `<sys/mman.h>`

TEST: The constants MCL_CURRENT and MCL_FUTURE are defined and are bitwise distinct.

ELSE *NO_OPTION*

Conformance for mlockall: PASS, NO_OPTION

10 **IF** *PCTS_mlockall* **THEN**

IF: *PCTS_GAP_mlockall* **THEN**

TEST: When the flag MCL_CURRENT is set, the call *mlockall (flags)* locks all of the pages currently mapped into the address space of the process.

TR: Test for at least two disjoint sets of pages.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

10 **IF** *PCTS_mlockall* **THEN**

IF: *PCTS_GAP_mlockall* **THEN**

TEST: When the flag MCL_FUTURE is set, the call *mlockall (flags)* locks all of the pages that become mapped into the address space of the process in the future, when those mappings are established.

TR: Test for at least two disjoint sets of pages.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_mlockall* **THEN**

TEST: The PCD.1b that documents the behavior if MCL_FUTURE is specified, and any of

7. the automatic locking of future mappings eventually causes the amounts of locked memory to exceed the amount of available physical memory
8. the automatic locking of future mappings eventually causes the amount of locked memory to exceed any other implementation-defined limit,
9. the manner in which the implementation informs the application of these situations

in §12.1.1.2.

ELSE *NO_OPTION*

Conformance for mlockall: PASS, NO_OPTION

munlockall

IF *PCTS_munlockall* **THEN**
TEST: A successful call to *munlockall*() unlocks all currently mapped pages of the address space of the process, with respect to the process's address space, and returns zero.
TR: Test for at least two disjoint sets of pages.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for munlockall: PASS, NO_OPTION

12 **IF** *PCTS_munlockall* **THEN**
IF: *PCTS_GAP_mlockall* **THEN**
TEST: Any pages that become mapped into the address space of the process after a call to *munlockall*() are not locked, unless there is an intervening call to *mlockall*() specifying MCL_FUTURE or a subsequent call to *mlockall*() specifying MCL_CURRENT.
TR: Test for both MCL_FUTURE and MCL_CURRENT.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_munlockall* **THEN**
TEST: When pages mapped into the address space of the process are also mapped into the address spaces of other processes, and are locked by those processes, the locks established by the other processes are unaffected by a call by this process to *munlockall*().
TR: Test for at least two other processes.
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_OPTION

14 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_GAP_munlockall* **THEN**
TEST: After a successful call to *mlockall*( ) that specifies MCL_CURRENT, all currently mapped pages of the process's address space are memory resident and locked.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_mlockall* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the memory residency of unlocked pages does so in §12.1.1.2.
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_OPTION

15 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_RAP_mlockall* **THEN**
TEST: Appropriate privilege is required to lock process memory with *mlockall*( ).
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

D_3 **IF** *PCTS_mlockall* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mlockall*( ) function does so in §12.1.1.2.
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_OPTION

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

TEST: A PCD.1b that documents whether or not it supports the *munlockall*( ) function does so in §12.1.1.2.

ELSE *NO_OPTION*

Conformance for mlockall: PASS, NO_OPTION

12.1.1.3 Returns

R_1 **IF** *PCTS_mlockall* **THEN**

TEST: When a call to *mlockall*() completes successfully, the interface returns a value of 0.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion mlockall in §12.1.1.2

R_2 **IF** *PCTS_mlockall* **THEN**

TEST: When a call to *mlockall*() completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and no additional memory is locked.

ELSE *NO_OPTION*

SEE: All assertions in §12.1.1.4 controlled by a *PCTS_mlockall*

D_5 **IF** *PCTS_mlockall* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the effect of failure of *mlockall*() on previously existing locks in the address space does so in §12.1.1.3.

R_3 **IF** *PCTS_mlockall* and a PCD.1b documents the following *PCTS_munlockall* **THEN**

TEST: The interface *munlockall*() returns a value of 0

ELSE *NO_OPTION*

SEE: Assertion munlockall in §12.1.1.2

12.1.1.4 Errors

16 **IF** not *PCTS_mlockall* **THEN:**

TEST: A call to *mlockall*(), returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for mlockall: PASS, NO_OPTION

17 **IF** not *PCTS_munlockall* **THEN:**

TEST: A call to *munlockall*(), returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for munlockall: PASS, NO_OPTION

18 **IF** *PCTS_mlockall* **THEN:**

IF *PCTS_GAP_mlockall* **THEN:**

TEST: A call to *mlockall*(),when some or all of the memory identified by the operation could not be locked, returns a value of -1 and sets *errno* to [EAGAIN].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_mlockall* **THEN:**

IF *PCTS_GAP_mlockall* **THEN:**

TEST: A call to *mlockall*(),when the *flags* argument is zero, returns a value of -1 and sets *errno* to [EINVAL].

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

20 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_GAP_mlockall* **THEN:**
TEST: A call to *mlockall*(),when the *flags* includes unimplemented flags, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns a *flags* argument that includes unimplemented flags or indicates that there is no way to generate a *flags* argument that includes unimplemented flags on the system.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_GAP_mlockall*
PCTS_DETECT_LOCKABLE_MEMORY_LIMITS_mlockall **THEN:**
TEST: A call to *mlockall*(),when locking all of the pages currently mapped into the address space of the process would exceed an implementation-defined limit on the amount of memory that the process may lock, , returns a value of -1 and sets *errno* to [ENOMEM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

D_6 **IF** *PCTS_mlockall* **THEN**
TEST: The PCD.1b documents the maximum amount of memory that process may lock in §12.1.1.4.
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_OPTION

22 **IF** *PCTS_mlockall* **THEN**
IF: *PCTS_RAP_mlockall PCTS_DETECT_NO_AP* **THEN:**
TEST: A call to *mlockall*(),when the calling process does not have the appropriate privilege to perform the requested operation, returns a value of -1 and sets *errno* to [EPERM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlockall: PASS, NO_TEST_SUPPORT, NO_OPTION

12.1.2 Lock/Unlock a Range of Process Address Space

Functions: *mlock*(), *munlock*()

12.1.2.1 Synopsis

1

M_GA_stdC_proto_decl(int; mlock; const void *addr, size_tlen; sys/mman.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for mlock: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(mlock; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for mlock: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; mlock; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sem_init: PASS, NO_OPTION

4

M_GA_macro_args (mlock; sys/mman.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for mlock: PASS, NO_OPTION

5

M_GA_stdC_proto_decl(int; munlock; const void *addr, size_tlen; sys/mman.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for munlock: PASS[5, 6], NO_OPTION

6

M_GA_commonC_int_result_decl(munlock; const void *addr, size_tlen; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for munlock: PASS[5, 6], NO_OPTION

7

M_GA_macro_result_decl(int; munlock; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for munlock: PASS, NO_OPTION

8

M_GA_macro_args (munlock; sys/mman.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for munlock: PASS, NO_OPTION

12.1.2.2 Description

mlock **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *execlp*() and *execvp*()
IF *PCTS_MLOCK* **THEN**
IF *PCTS_GAP_mlock* **THEN**
TEST: A successful call to the function *mlock* (*addr, len*) returns a value of zero and causes those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes to be memory resident until unlocked or until the process exits or executes a successful call to *function*().
TR: Test for each of the three conditions.
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** a *PCTS_mlock* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the argument *addr*.,in a call to *mlock (addr, len)*, must be a multiple of the page size {PAGESIZE} does so in §12.1.2.2.
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

munlock
IF *PCTS_ munlock* **THEN**
IF *PCTS_mlock* and *PCTS_GAP_mlock* **THEN**
TEST: The call *munlock (addr, len)* unlocks those whole pages containing any part of the address space of the process, with respect to the address space of the process, starting at address *addr* and continuing by *len* bytes, regardless of how many times *mlock*() has been called by the process for any of the pages in the specified ranges, and returns zero.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for munlock: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_munlock* documents the following **THEN**
TEST: A PCD.1b that documents the argument *addr*.,in a call to *munlock (addr, len)*, must be a multiple of the page size {PAGESIZE} DOES so in §12.1.2.2.
ELSE *NO_OPTION*
Conformance for munlock: PASS, NO_OPTION

9 **IF** *PCTS_munlock* **THEN**
TEST: When any of the pages in the range specified to a call to *munlock*() are also mapped into the address spaces of other processes, any locks established on those pages by another process are unaffected by the call of this process to *munlock*().
ELSE *NO_OPTION*
Conformance for munlock: PASS, NO_OPTION

10 **IF** *PCTS_munlock* **THEN**
TEST: When any of the pages in the range specified to a call to *munlock*() are also mapped into the address spaces of other processes, any locks established on those pages via the other mappings are unaffected by this call.
ELSE *NO_OPTION*
Conformance for munlock: PASS, NO_OPTION

D_3 **IF** *PCTS_mlock* and PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents memory residency of unlocked pages does so in §12.1.2.2.
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

11 **IF** *PCTS_ mlock* **THEN**
IF *PCTS_RAP_mlock* **THEN**
TEST: Appropriate privilege is required to lock process memory with *mlock*().
ELSE *NO_TEST_SUPPORT*

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

D_4 **IF** *PCTS_mlock* and PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mlock*() function does so in §12.1.2.2.
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

D_5 **IF** *PCTS_munlock* and PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *munlock*() does so in §12.1.2.2.
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

12.1.2.3 Returns

R_1 **IF** *PCTS_mlock* **THEN**
TEST: When a call to *mlock*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion mlock in §12.1.2.2

R_2 **IF** *PCTS_mlock* **THEN**
TEST: When a call to *mlock*() completes successfully, the interface returns a value of -1, sets *errno* to indicate the error, and no change is made to any locks in the address space.
PCTS_mlock
ELSE *NO_OPTION*
SEE: All assertions in §12.1.2.4 controlled by *PCTS_mlock*

R_3 **IF** *PCTS_munlock* **THEN**
TEST: When a call to *munlock*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion munlock in §12.1.2.2

R_4 **IF** *PCTS_munlock* **THEN**
TEST: When a call to *munlock*() completes unsuccessfully, the interface returns a value of -1, sets *errno* to indicate the error, and no change is made to any locks in the address space.
PCTS_munlock
ELSE *NO_OPTION*
SEE: All assertions in §12.1.2.4 controlled by *PCTS_munlock*

12.1.2.4 Errors

12 **IF** *PCTS_mlock* **THEN**
IF: *PCTS_GAP_mlock* **THEN**
TEST: When a call to *mlock*(), when some or all of the address range specified by the *addr* and *len* arguments does not correspond to valid mapped pages in the

address space of the process, returns a value of -1 and sets *errno* to [ENOMEN].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_munlock* **THEN**

TEST: When a call to *munlock*(), when some or all of the address range specified by the *addr* and *len* arguments does not correspond to valid mapped pages in the address space of the process, returns a value of -1 and sets *errno* to [ENOMEN].

ELSE *NO_OPTION*

Conformance for munlock: PASS, NO_OPTION

14 **IF** not *PCTS_mlock* **THEN**

TEST: A call to *mlock*(), returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_OPTION

15 **IF** not *PCTS_munlock* **THEN**

TEST: A call to *munlock*(), returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_OPTION

16 **IF** *PCTS_mlock* **THEN**

IF: *PCTS_GAP_mlock* **THEN**

TEST: A call to *mlock*(), when some or all of the memory identified by the operation could not be locked, returns a value of -1 and sets *errno* to [EAGAIN].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

D_6 **IF** *PCTS_mlock* or *PCTS_munlock* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the implementation requires memory locking only in multiples of {PAGESIZE} does so in §12.1.2.4.

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_OPTION

17 **IF** *PCTS_mlock* and *PCTS_MULTIPLE_OF_PAGESIZE* and *PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE* **THEN**

IF *PCTS_GAP_mlock* **THEN**

TEST: A call to *mlock*(), when the *addr* argument is not a multiple of the page size {PAGESIZE}, returns a value of -1 and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_MULTIPLE_OF_PAGESIZE* and *PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE* and *PCTS_munlock* **THEN**

TEST: A call to *munlock*(), when the *addr* argument is not a multiple of the page size {PAGESIZE}, returns a value of -1 and sets *errno* to [EINVAL].

ELSE *NO_OPTION*

Conformance for mlock: PASS,, NO_OPTION

D_7 **IF** *PCTS_mlock* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the implementation defined limit on an amount of memory that a process may lock does so in §12.1.2.4.
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

19 **IF** *PCTS_ mlock* and *PCTS_DETECT_LOCKABLE_MEMORY_LIMIT_ mlock* **THEN**
IF *PCTS_GAP_mlock* **THEN**
TEST: A call to *mlock*(), when locking the pages mapped by the specified range would exceed an implementation-defined limit on the amount of memory that the process may lock, returns a value of -1 and sets *errno* to [ENOMEM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

D_8 **IF** *PCTS_mlock* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the implementation detects whether or not a process has appropriate privileges to lock pages does so in §12.1.2.4.
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_OPTION

20 **IF** *PCTS_ mlock* and *PCTS_DETECT_NO_AP* **THEN**
IF *PCTS_RAP_mlock* **THEN**
TEST: A call to *mlock*(), when the calling process does not have the appropriate privilege to perform the requested operation , returns a value of -1 and sets *errno* to [EPERM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mlock: PASS, NO_TEST_SUPPORT, NO_OPTION

12.2 Memory Mapping Functions

12.2.1 Map Process Address to a Memory Object

Function: *mmap*()

1

M_GA_stdC_proto_decl(void *; mmap; void*addr, size_tlen; int prot, int flags, int fildes, off_t off; sys/mman.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for mlock: PASS[1, 2], NO_OPTION

2

M_GA_commonC_result_decl (void*; mmap; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for mlock: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; mlock; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sem_init: PASS, NO_OPTION

4

M_GA_macro_args (mmap; sys/mman.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for mlock: PASS, NO_OPTION

mmap **IF** *PCTS_mmap* **THEN**
TEST: A successful call to the function *mmap*() establishes a mapping between the address space of the process for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off* for *len* bytes, and returns the address at which the mapping was placed.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

D_1 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the implementation-dependent function of the parameter *addr* and the values of *flags*, which determines the address at which the mapping is placed, does so in §12.1.2.2.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

5 **IF** *PCTS_mmap* **THEN**
TEST: The address range starting at the address returned by *mmap(addr, len, prot, flags, fildes, off),* and continuing for *len* bytes, is legitimate for the possible (not necessarily current) address space of the process.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

6 **IF** *PCTS_mmap* **THEN**
TEST: In a call to *mmap(addr, len, prot, flags, fildes, off)* the ranges of bytes starting at *off* and continuing for *len* bytes is legitimate for the possible (not necessarily current) offsets in the file or shared memory object represented by *fildes*..
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

7 **IF** *PCTS_mmap* **THEN**
TEST: The mapping established by *mmap(addr, len, prot, flags, fildes, off),*replaces any previou7s mappings for those whole pages containing any part of the address space of the process, starting at the address returned by *mmap*() and continuing for *len* bytes.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

8 **IF** *PCTS_mmap* **THEN**

776 **TEST**: In a cal 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 artument *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**

TEST: The implementation supports the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the inclusive OR of PROT_READ and PROT_WRITE

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

12 **IF** *PCTS_ mmap* **THEN**

TEST: after a call ti *mmap(addr, len, prot, flags, fildes, off)* with the flag MAP_SHARED SET, write references change the underlying object.

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_ mmap* **THEN**

IF *PCTS_MAP_PRIVATE* **THEN**

TEST: After a call to *mmap(addr, len, prot, flags, fildes, off),* with the flag MAP_PRIVATE set, modifications to the mapped data by the calling process are visibly only to the calling process and do not change the underlying object.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether modifications to the underlying object, done after the MAP_PRIVATE mapping is established, are visible through the MAP_PRIVATE mapping, does so in §12.1.2.2.

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

14 **IF** *PCTS_ mmap* **THEN**

IF *PCTS_MAP_PRIVATE* **THEN**

TEST: MAP_SHARED must be specified in a call to the *mmap*() function.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_ mmap* **THEN**

TEST: the mappint type is retained across *fork*().

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

16 **IF** *PCTS_ mmap* **THEN**

IF *PCTS_MAP_FIXED* **THEN**

TEST: When MAP_FIXED is set, the address returned by *mmap(addr, len, prot, flags, fildes, off)* is *addr* exactly..

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

D_3 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether MAP_FIXED is supported in §12.1.2.2.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

D_4 **IF** *PCTS_mmap* **THEN**
TEST: A PCD.1b documents how the system uses *addr* to arrive at *pa*, when MAP_FIXED is not set in §12.1.2.2.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

17 **IF** *PCTS_ mmap* **THEN**
TEST: When MAP_FIXED is not set, a call to *mmap()* never places a mapping at address zero.
ELSE *NO_OPTION*
Conformance for mmap: PASS,, NO_OPTION

18 **IF** *PCTS_ mmap* **THEN**
TEST: When MAP_FIXED is not set, a call to *mmap()* never replaces an extant mapping.
ELSE *NO_OPTION*
Conformance for mmap: PASS,, NO_OPTION

19 **IF** *PCTS_ mmap* **THEN**
IF *PCTS_MAP_FIXED* **THEN**
TEST: When MAP_FIXED is specified and *addr* is nonzero, the address returned by *mmap(addr, len, prot, flags, fildes, off)* has the same remainder as the *off* parameter, modulo the page size {PAGESIZE}.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

20 **IF** *PCTS_ mmap* **THEN**
IF *PCTS_MULTIPLE_OF_PAGESIZE* **THEN**
TEST: The argument *off* must be a multiple of the page size..
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

D_5 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents that the argument *off* , in a call to *mmap(addr, len, prot, flags, fildes, off)* must be a multiple of the page size, does so in §12.1.2.2.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

21 **IF** *PCTS_ mmap* **THEN**
IF *PCTS_MULTIPLE_OF_PAGESIZE* and *PCTS_MAP_FIXED* **THEN**
TEST: When MAP_FIXED is specified, the argument *addr* must be a multiple of the page size.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

D_6 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents that the argument *addr* , in a call to *mmap(addr, len, prot, flags, fildes, off)* with MAP_FIXED specified, must be a multiple of the page size, does so in §12.1.2.2.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

22 **IF** *PCTS_ mmap* **THEN**
IF *PCTS_MAP_FIXED* **THEN**
TEST: When MAP_FIXED is specified, the parameter *len* need not meet a size or alignment constraint.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

23 **IF** *PCTS_ mmap* **THEN**
IF *PCTS_MAP_FIXED* **THEN**
TEST: When MAP_FIXED is specified, after a call to *mmap(addr, len, prot, flags, fildes, off)* any partial page specified by the address range starting at the returned address and continuing for *len* bytes is included in the object.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

24 **IF** *PCTS_ mmap* **THEN**
TEST: Any partial page at the end of an object is zero-filled.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

25 **IF** *PCTS_ mmap* **THEN**
TEST: Modified portions of the last page of an object that are beyond its end are not written out.
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

mmap_SIGBUS
IF *PCTS_ mmap* **THEN**
IF {POSIX_MEMORY_PROTECTION} **THEN**
TEST: Following a call to *mmap(addr, len, prot, flags, fildes, off)*, references within the address range starting at the returned address

and continuing for *len* bytes to whole pages following the end of an object result in delivery of a SIGBUS signal.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

D_7 **IF** *PCTS_mmap* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the result of references within the address range starting at *pa* and continuing for *len* bytes, to whole pages following the end of an object, when the {_POSIX_MEMORY_PROTECTION} option is not supported, does so in §12.1.2.2.

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

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

TEST: A PCD.1b that documents whether or not it supports the *mmap*() function, does so in §12.1.2.2.

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

12.2.1.3 Returns

R_2 **IF** *PCTS_mmap* **THEN**

TEST: When a call to *mmap( )* completes successfully, the interface returns the address at which the mapping was placed.

ELSE *NO_OPTION*

SEE: Assertion mmap in §12.2.1.2

R_3 **IF** *PCTS_mmap* **THEN**

TEST: When a call to *mmap( )* completes unsuccessfully, the interface returns a value of MAP_FAILED, and sets *errno* to indicate the error

ELSE *NO_OPTION*

SEE: Assertion mmap in §12.2.1.4

26 **IF** *PCTS_ mmap* **THEN**

SETUP: Include the header `<sys/mman.h>`

TEST: The symbol MAP_FAILED is defined.

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

R_4 **IF** *PCTS_mmap* **THEN**

TEST: No successful return from *mmap*() returns the value of MAP_FAILED.

ELSE *NO_OPTION*

SEE: Assertion mmap in §12.2.1.4

12.2.1.4 Errors

27 **IF** *PCTS_ mmap* **THEN**

TEST: A call to *mmap(addr, len, prot, flags, fildes, off*), when the file descriptor *fildes* is not open for read, returns a value of MAP_FAILED and sets *errno* to [EACCESS].

TR: Specify each possible protection

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

28 **IF** *PCTS_ mmap* **THEN**

TEST: A call to *mmap(addr, len, prot, flags, fildes, off*), when the file descriptor *fildes* is not open for write and PROT_WRITE is specified for a MAP_SHARED type mapping, returns a value of MAP_FAILED and sets *errno* to [EACCESS].

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

29 **IF** *PCTS_ mmap* **THEN**

TEST: A call to *mmap()*, when the mapping could not be locked in memory, if required by *mlockall*(), due to a lack of resources, returns a value of MAP_FAILED and sets *errno* to [EAGAIN].

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

30 **IF** *PCTS_ mmap* **THEN**

TEST: A call to *mmap(addr, len, prot, flags, fildes, off*), when the *fildes* argument is not a valid open file descriptor, returns a value of MAP_FAILED and sets *errno* to EBADF].

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

31 **IF** *PCTS_ mmap* **THEN**

IF *PCTS_DETECT_INVALID_FLAGS_mmap* **THEN**

TEST: Following a call to *mmap(addr, len, prot, flags, fildes, off)*, when the value in *flags* is invalid (e.g., neither MAP_PRIVATE or MAP_SHARED is set) returns a value of MAP_FAILED and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

32 **IF** *PCTS_ mmap* **THEN**

TEST: A call to *mmap(addr, len, prot, flags, fildes, off*), when the *fildes* argument refers to an object for which *mmap*() is meaningless, such as a terminal, returns a value of MAP_FAILED and sets *errno* to [ENODEV].

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_OPTION

33 **IF** *PCTS_ mmap* **THEN**

IF *PCTS_MAP_FIXED* **THEN**

TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when MAP_FIXED is specified, and the address range starting at *addr* and continuing for *len* bytes exceeds that allowed for the address space of a process; or MAP_FIXED is not specified, and there is insufficient room in the address space to effect the mapping; returns a value of MAP_FAILED and sets *errno* to [ENOMEM].
There is no known reliable test method for this assertion.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

34 **IF** *PCTS_ mmap* **THEN**

IF *PCTS_mlockall PCTS_GAP_mlockall* **THEN**

TEST: A call to *mmap()*, when the mapping could not be locked in memory, if required by *mlockall*(), because it would require more space than the system is able to supply, returns a value of MAP_FAILED and sets *errno* to [ENOMEM].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

35 **IF** not *PCTS_ mmap* **THEN**

TEST: A call to *mmap()*, returns a value of MAP_FAILED and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for mmap: PASS,, NO_OPTION

36 **IF** *PCTS_ mmap* **THEN**

IF not *PCTS_MAP_FIXED* **THEN**

TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when MAP_FIXED is specified in the *flags* argument, returns a value of MAP_FAILED and sets *errno* to [ENOTSUP].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

37 **IF** *PCTS_ mmap* **THEN**

IF not *PCTS_MAP_PRIVATE* **THEN**

TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when MAP_PRIVATE is specified in the *flags* argument, a value of MAP_FAILED and sets *errno* to [ENOTSUP].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mmap: PASS, NO_TEST_SUPPORT, NO_OPTION

mmap_ENOTSUP

IF *PCTS_ mmap* **THEN**

TEST: A call to *mmap()*, when the implementation does not support the combination of accesses requested in the *prot* argument, returns a value of MAP_FAILED and sets *errno* to [ENOTSUP].
TR: Test with MAP_SHARED and MAP_PRIVATE both in the *prot* argument.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS,, NO_OPTION

38 **IF** *PCTS_ mmap* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when the addresses in the range starting at *off* and continuing for *len* bytes are invalid for the object specified by *fildes*, returns a value of MAP_FAILED and sets *errno* to [ENXIO].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_OPTION

39 **IF** *PCTS_ mmap* **THEN**
IF *PCTS_MAP_FIXED* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when MAP_FIXED is specified in *flags* and the combination of *addr*, *len*, and *off* is invalid for the object specified by *fildes*, returns a value of MAP_FAILED and sets *errno* to [ENXIO].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST SUPPORT, NO_OPTION

40 **IF** *PCTS_ mmap* **THEN**
IF *PCTS_MULTIPLE_OF_PAGESIZE* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when the argument *off* is not a multiple of the page size {PAGESIZE}, returns a value of MAP_FAILED and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_ SUPPORT, NO_OPTION

41 **IF** *PCTS_ mmap* **THEN**
IF *PCTS_MULTIPLE_OF_PAGESIZE* and *PCTS_MAP_FIXED* **THEN**
TEST: A call to *mmap(addr, len, prot, flags, fildes, off)*, when MAP_FIXED is specified and the argument *addr* is not a multiple of the page size {PAGESIZE}, returns a value of MAP_FAILED and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mmap: PASS, NO_TEST_ SUPPORT, NO_OPTION

12.2.2.2 Unmap Previously Mapped Addresses

function: *munmap*()

12.2.2.1 Synopsis

1

M_GA_stdC_proto_decl(int;; munmap; void*addr, size_t len; sys/mman.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for munmap: PASS[1, 2], NO_OPTION

2

M_GA_commonC_result_decl (munmap; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for munmap: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; munmap; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for munmap: PASS, NO_OPTION

4

M_GA_macro_args (munmap; sys/mman.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for munmap: PASS, NO_OPTION

12.2.2.2 Description

munmap

IF *PCTS_ munmap* **THEN**
TEST: A successful call to the function *munmap(addr, len)*, removes any mappings for those entire pages containing any part of the address space of the process starting at *addr* and continuing for *len* bytes, and returns the value zero.
ELSE *NO_OPTION*
Conformance for munmap: PASS, NO_OPTION

munmap_SIGSEV

IF *PCTS_ munmap* **THEN**
TEST: Following a successful call to *munmap()*, references to the unmapped pages results in the delivery of a SIGSEV signal to the process.
ELSE *NO_OPTION*
Conformance for munmap: PASS, NO_OPTION

5 **IF** *PCTS_ munmap* **THEN**
TEST: When there are no mappings in the specified address range, then *munmap()* has no effect.
ELSE *NO_OPTION*
Conformance for munmap: PASS, NO_OPTION

6 **IF** *PCTS_ munmap* **THEN**
IF *PCTS_MULTIPLE_OF_PAGESIZE* **THEN**

TEST: In a call to *munmap(addr, len)*, the argument *addr* must be a multiple of the page size, {PAGESIZE}.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_TEST_SUPPORT, NO_OPTION

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

TEST: A PCD.1b that documents that the argument *addr*, in a call to *munmap*(*addr, len*), must be a multiple of the page size, {PAGESIZE}, does so in §12.2.2.2.

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_OPTION

7 **IF** *PCTS_ munmap* **THEN**

IF *PCTS_munmap* and *PCTS_MAP_PRIVATE* **THEN**

TEST: When a mapping to be removed is private, any modifications made in this address range are discarded.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_TEST_SUPPORT, NO_OPTION

munlock_remove_maps

IF *PCTS_ munmap* **THEN**

IF *PCTS_mlock* and *PCTS_GAP_MLOCK* and *PCTS_mlockall* and *PCTS_GAP_MLOCKALL and PCTS_munlock* **THEN**

TEST: Following a call to *munmap(addr, len,)*, any memory locks (see POSIX.1b {3} §12.1 2 and POSIX.1b {3} §12.1 1 is associated with this address range are removed, as if by an appropriate call to *munlock*().

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_TEST_ SUPPORT, NO_OPTION

D_2 **IF** *PCTS_munmap* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the behavior of this function if the mapping was not established by a call to *munmap*(), does so in §12.2.2.2.

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_OPTION

D_3 **IF** *PCTS_munmap* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *munmap*() function does so in §12.2.2.2.

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_OPTION

12.2.2.3 Returns

R_1 **IF** *PCTS_munmap* **THEN**

TEST: When a call to *munmap*() completes successfully, the interface returns to a value of 0.

ELSE *NO_OPTION*

SEE: Assertion munmap in §12.2.2.2

R_2 **IF** *PCTS_munmap* **THEN**

TEST: When a call to *munmap*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: All assertions in §12.2.2.4

12.2.2.4 Errors

8 **IF** *PCTS_ munmap* **THEN**

TEST: A call to *munmap*(), when some of the addresses in the range starting at *addr* and continuing for *len* bytes are outside the range allowed for the address space of a process, returns a value of -1 and sets *errno* to [EINVAL].

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

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_TEST, NO_OPTION

9 **IF** not *PCTS_ munmap* **THEN**

TEST: A call to *munmap*(), returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_OPTION

10 **IF** *PCTS_ munmap* **THEN**

IF *PCTS_MULTIPLE_OF_PAGESIZE*
and *PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE* **THEN**

TEST: A call to *munmap(addr, len,)*, when the value of *addr* is not a multiple of the page size {PAGESIZE}, returns a value of -1 and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for munmap: PASS, NO_TEST_ SUPPORT, NO_OPTION

12.2.3 Change Memory Protection

Function: *mprotect*()

12.2.3.1 Synopsis

1

M_GA_stdC_proto_decl(int; mprotect; const void*addr, size_t len; int_prot; sys/mman.h;;;)

SEE: Assertion GA_stdC_proto_decl in §2.7.3

Conformance for mprotect: PASS[1, 2], NO_OPTION

2

M_GA_commonC_result_decl (mprotect; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for munmap: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; mprotect; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for mprotect: PASS, NO_OPTION

4

M_GA_macro_args (mprotect; sys/mman.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for mprotect: PASS, NO_OPTION

12.2.3.2 Description

mprotect

IF *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A successful call to the function *mprotect(addr, len, prot)* changes the access protections to be that specified by *prot* for those whole pages containing any part of the address space of the process, starting at address *addr* and continuing for *len* bytes, and returns the value zero.
TR: Test for PROT_READ, PROT_WRITE, and PROT_NONE individually.
ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

R-1 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*(). Include the header `sys/mman.h>.`
TEST: The constants PROT_NONE.PROT_READ, PROT_WRITE, and PROT_EXEC are defined and are bitwise distinct.
ELSE *NO_OPTION*
SEE: Assertion prot_values in §12.2.1.2
Conformance for mprotect: PASS,, NO_OPTION

5 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: In the call *mprotect(addr, len, prot)*, the only permitted values for *prot* are PROT_NONE or the bitwise inclusive OR of one or more of the values PROT_READ, PROT_WRITE.
ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

R-2 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When an implementation cannot support the combination of access types specified by *prot*, the call to *mprotect*() fails.
ELSE *NO_OPTION*
SEE: Assertion mprotect_ENOTSUP in §12.2.3.4

D_1 **IF** *PCTS_mprotect* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether accesses other than those specified by *prot* does so in §12.2.3.2.

ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

6 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: All accesses fail where PROT_WRITE alone has been set.
ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

7 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: All accesses fail where PROT_NONE alone has been set.
ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

8 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: The implementation supports at least the following values of *prot* PROT_NONE, PROT_READ, PROT_WRITE and the inclusive OR of PROT_READ, and PROT_WRITE.
ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

9 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When MAP_PRIVATE was not specified in the original mapping, and PROT_WRITE is specified, mapped objects are opened in the specified address range with write permission.
TR: Try with and without closing file descriptors used to map the objects after performing the original mapping.
ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

D_2 **IF** *PCTS_mprotect* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the argument *addr* in a call to *mprotect(addr, len, prot)*, must be a multiple of the page size {PAGESIZE}, does so in §12.2.3.2.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

D_3 **IF** *PCTS_mprotect* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of this function if the mapping was not established by a call to *mmap*(), does so in §12.2.3.2.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

D_4 **IF** *PCTS_mprotect* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mprotect*() function does so in §12.2.3.2.
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_OPTION

12.2.3.3 Returns

R_3 IF *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When a call to *mprotect*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion mprotect in §12.2.3.2

R_4 IF *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When a call to *mprotect*() completes unsuccessfully, the interface returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §12.2.3.4

10 IF *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: When *mprotect*() fails and returns [EINVAL], the protections in the pages in the address range starting at *addr* and continuing for *len* bytes are unchanged.
ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

12.2.3.4 Errors

11 IF *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect*(), when the memory object was not opened for read, regardless of the protection specified, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

12 IF *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect(addr, len, prot)*, when the memory object was not opened for write, and PROT_WRITE is specified for a MAP_SHARED type mapping, returns a value of -1 and sets *errno* to [EACCESS]..
ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

13 IF *PCTS_mprotect* and *PCTS_mmap* **THEN**
IF *PCTS_MAP_PRIVATE* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect(addr, len, prot)*, when the *prot* argument specifies PROT_WRITE on a MAP_PRIVATE mapping, and there are insufficient memory resources to reserve for locking the private pages, if required, returns a value of -1 and sets *errno* to [EAGAIN].
NOTE: There is no known reliable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_TEST, NO_TEST_ SUPPORT, NO_OPTION

14 IF *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect(addr, len, prot)*, when the addresses in the range starting at *addr* and continuing for *len* bytes are outside the range allowed for the address space of a process returns a value of -1 and sets *errno* to [ENOMEM].

ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

15 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect(addr, len, prot)*, when the addresses in the range starting at *addr* and continuing for *len* bytes specify one or more pages that are not mapped, returns a value of -1 and sets *errno* to [ENOMEM].
ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

16 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
IF *PCTS_MAP_PRIVATE* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect(addr, len, prot)*, when the *prot* argument specifies PROT_WRITE on a MAP_PRIVATE mapping, and it would require more space than the system is able to supply for locking the private pages, if required, returns a value of -1 and sets *errno* to [ENOMEM].

ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_TEST, NO_TEST_ SUPPORT, NO_OPTION

17 **IF** not *PCTS_mprotect* **THEN**
TEST: A call to *mprotect()* returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mprotect: PASS,, NO_OPTION

mprotect_ENOTSUP
IF *PCTS_mprotect* and *PCTS_mmap* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect()*, when the implementation does not support the combination of accesses requested in the *prot* argument, returns a value of -1 and sets *errno* to [ENOTSUP].
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_mprotect* and *PCTS_mmap* **THEN**
IF *PCTS_MULTIPLE_OF_PAGESIZE*
and *PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE* **THEN**
SETUP: Map memory pages using the function *mmap*().
TEST: A call to *mprotect(addr, len, prot)*, when the value of *addr* is not a multiple of the page size {PAGESIZE}, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mprotect: PASS, NO_TEST_ SUPPORT, NO_OPTION

12.2.4 Memory Object Synchronization

Function: *msync*()

12.2.4.1 Synopsis

1

M_GA_stdC_proto_decl(int; msync; void*addr, size_t len; int_flags; sys/mman.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for msync: PASS[1, 2], NO_OPTION

2

M_GA_commonC_result_decl (msync; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for msync: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; msync; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for msync: PASS, NO_OPTION

4

M_GA_macro_args (msync; sys/mman.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for msync: PASS, NO_OPTION

12.2.4.2 Description

msync **IF** *PCTS_msync* and *PCTS_munmap* **THEN**
IF: *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: A successful call to the function *msync*() writes all modified data to permanent storage locations in those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes and returns the value zero.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_msync* and *PCTS_munmap* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: A successful call to the function *msync*(*addr, len, flags*) where the *flags* argument contains the value MS_INVALIDATE invalidates cached copies of the data.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST, NO_OPTION

D_1 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents a call to *msync*(*addr, len, flags*), when there are no permanent storage locations to write the modified data does so in §12.2.4.2.
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

D_2 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents that the argument *addr*, in a call to *msync*(*addr, len, flags*), must be a multiple of the page size {PAGESIZE}, does so in §12.2.4.2.

ELSE *NO_OPTION*

Conformance for msync: PASS, NO_OPTION

D_3 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether the implementation also writes out other file attributes does so in §12.2.4.2.

ELSE *NO_OPTION*

Conformance for msync: PASS, NO_OPTION

6 **IF** *PCTS_msync* and *PCTS_munmap* **THEN**

IF *PCTS_msync_storage* and *PCTS_MAP_PRIVATE* **THEN**

SETUP: Map memory pages using the function *mmap*()

TEST: When the *msync*() function is called on MAP_PRIVATE mapping, any modified data is not written to the underlying object and does not cause such data to be made visible to other processes.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for msync: PASS, NO_TEST, NO_OPTION

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

TEST: A PCD.1b that documents whether data in MAP_PRIVATE mappings has any permanent storage locations does so in §12.2.4.2.

ELSE *NO_OPTION*

Conformance for msync: PASS, NO_OPTION

D_5 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the effect of *msync*() on shared memory objects §12.2.4.2.

ELSE *NO_OPTION*

Conformance for msync: PASS, NO_OPTION

7 **IF** *PCTS_msync* **THEN**

SETUP: Include the header `sys/msync.h>`.

TEST: The constants MS_ASYNC, MS_SYNC, and MS_INVALIDATE are defined and are bitwise distinct.

ELSE *NO_OPTION*

Conformance for msync: PASS,, NO_OPTION

8 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**

IF *PCTS_msync_storage* **THEN**

SETUP: Map memory pages using the function *mmap*()

TEST: When MS_ASYNC the *msync*() returns immediately, once all the write operations are initiated or queued for servicing.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for msync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**

IF *PCTS_msync_storage* **THEN**

SETUP: Map memory pages using the function *mmap*()

TEST: When MS_SYNC *msync*() does not return until all write operations are completed as defined for synchronized I/O data integrity completion.

NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWrite in §2.2.119
Conformance for msync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

10 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: When MS_ASYNC is specified, *msync*() all write operations are completed as defined for synchronized I/O data integrity completion.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion GA_syncIODataIntegrityWrite in §2.2.119
Conformance for msync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

R_1 **IF** *PCTS_msync* **THEN**
TEST: A call to *msync*() cannot specify both MS_ASYNC and MS_SYNC
ELSE *NO_OPTION*
SEE: Assertions msync_einval in §12.2.4.4

11 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: Following a call to *msync*(*addr, len, flags*) with MS_INVALIDATE set, references to the object obtain data that was consistent with the permanent storage locations sometime between the call to *msync*() and the first subsequent memory reference to the data.
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test..
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

D_6 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of this function, if the mapping was not established by a call to *mmap*(), does so in §12.2.4.2.
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

D_7 **IF** *PCTS_msync* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *msync*() function does so in §12.2.4.2.
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

12.2.4.3 Returns

R_2 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: When a call to *msync*() completes successfully, the interface returns a value of 0.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*
SEE: Assertion msync in §12.2.4.2

R_3 IF *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: When a call to *msync*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion msync in §12.2.4.4

12.2.4.4 Errors

12 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: A call to *msync*(*addr, len, flags*), when some or all of the address in the range starting at *addr* and continuing for *len* bytes are locked, and MS_INVALIDATE is specified, returns a value of -1 and sets *errno* to [EBUSY].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

msync_einval
IF *PCTS_msync* and *PCTS_mmap* **THEN**
IF: *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: A call to *msync*(*addr, len, flags*), when the value in *flags* is invalid, returns a value of -1 and sets *errno* to [EINVAL].
TR: Test with *flags* having both MS_ASYNC and MS_SYNC in it.
NOTE: A subroutine is recommended that either returns a *flags* argument that includes unimplemented flags or indicates that there is no way to generate a *flags* argument that includes unimplemented flags on the system.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: A call to *msync*(*addr, len, flags*), when the addresses in the range starting at *add* and continuing for *len* bytes are outside the range allowed for the address space of a process, returns a value of -1 and sets *errno* to [ENOMEM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

14 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: A call to *msync*(*addr, len, flags*), when the address in the range starting at *addr* and continuing for *len* specify one of more pages that are not mapped, returns a value of -1 and sets *errno* to [ENOMEM].
TR: Test for one page not mapped and for more than one page not mapped.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** not *PCTS_msync* **THEN**
TEST: A call to *msync*(), returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_OPTION

16 **IF** *PCTS_msync* and *PCTS_mmap* **THEN**
IF *PCTS_msync_storage* and *PCTS_MULTIPLE_OF_PAGESIZE* and *PCTS_DETECT_NOT_MULTIPLE_OF_PAGESIZE* **THEN**
SETUP: Map memory pages using the function *mmap*()
TEST: A call to *msync*(*addr, len, flags*), when the value of *addr* is not a multiple of the page size {PAGESIZE}, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for msync: PASS, NO_TEST_SUPPORT, NO_OPTION

12.3 Shared Memory Functions

12.3.1 Open a Shared Memory Object

Function: *shm_open*()

12.3.1.1 Synopsis

1

M_GA_stdC_proto_decl(int; shm_open; const char*name, int oflag, mod_t mode; sys/mman.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for shm_open: PASS[1, 2], NO_OPTION

2

M_GA_commonC_result_decl (shm_open; sys/mman.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for shm_open:: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; shm_open; sys/mman.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for shm_open: PASS, NO_OPTION

4

M_GA_macro_args (shm_open; sys/mman.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for shm_open: PASS, NO_OPTION

12.3.1.2 Description

shm_open
IF *PCTS_shm_open* **THEN**
TEST: A successful call to the *shm_open*() function creates an open file description that refers to the shared memory object, and returns a valid file descriptor that refers to that open file description.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

5 **IF** *PCTS_shm_open* **THEN**

TEST: The file descriptor returned by *shm_open*() is a nonnegative integer.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_1 **IF** *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the shared memory object *name* appears in the file system and is visible to other functions that take pathnames as arguments does so in §12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

6
M_GA_portableFilenames(shm_open)
SEE: Assertion GA_portableFilenames in §2.2.4.0
Conformance for shm_open: PASS, NO_OPTION

7
M_GA_upperLowerNames (shm_open)
SEE: Assertion GA_upperLowerNames in §2.2.2.40
Conformance for shm_open:: PASS, NO_OPTION

8
M_GA_PRNoTRUNC(shm_open)
SEE: Assertion GA_PRNoTrunc in §2.3.6
Conformance for shm_open: PASS, NO_OPTION

9
M_GA_PRNoTruncError (shm_open)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for shm_open: PASS, NO_OPTION

10 **IF** *PCTS_shm_open* **THEN**
TEST: When *name* begins with the slash character, then processes calling *shm_open*(*name, oflag, mode*), with the same value of *name* refer to the same shared memory object, as long as that name has not been removed.
TR: Test using two different processes.
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

D_2 **IF** *PCTS_shm_open* **THEN**
TEST: A PCD.1b that documents the effect if the *name* to *shm_open(name, oflag, mode)* does not begin with the slash character in §12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_3 **IF** *PCTS_shm_open* **THEN**
TEST: A PCD.1b that documents the interpretation of slash characters other than the leading slash character in the *name* argument to *shm_open(name, oflag, mode)* in §12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

11 **IF** *PCTS_shm_open* **THEN**
TEST: A successful call to *shm_open*(), returns a file descriptor for the shared memory object that is the lowest-numbered file descriptor not currently open for that process..
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

12 **IF** *PCTS_shm_open* **THEN**
TEST: The process does not share the open file description created by *shm_open*() with any other processes.
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

D_4 **IF** *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the file offset is set by *shm-open*()does so in §12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

13 **IF** *PCTS_shm_open* **THEN**
TEST: The FD_CLOEXEC file descriptor flag associated with the new file descriptor is set by *shm_open*()
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

14 **IF** *PCTS_shm_open* **THEN**
TEST: Following a call to *shm_open(name, oflag, mode)*, the file status flags and file access modes of the open file description are set according to the value of *oflag*.
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

15 **IF** *PCTS_shm_open* **THEN**
SETUP: Include the header `<sys/mman.h>`
TEST: The constants O_RDONLY, O_RDWR, O_CREAT, O_EXCL and O_TRUNC are defined, and have the same values as defined in the header `<fcntl.h>`
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

16 **IF** *PCTS_shm_open* **THEN**
SETUP: Include the header `<sys/mman.h>`
TEST In a call to *shm_open(name, oflag, mode)*, the *oflag* argument must be the bitwise inclusive OR of one or more of the following flags: O_RDONLY, O_RDWR, O_CREAT, O_EXCL and O_TRUNC**.**
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

17 **IF** *PCTS_shm_open* **THEN**
TEST: In the call to *shm_open(name, oflag, mode)*, *oflag* must have either O_RDONLY, or O_RDWR set**.**
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

18 **IF** *PCTS_shm_open* **THEN**
TEST: In the call to *shm_open(name, oflag, mode)*, *oflag* must have both O_RDONLY, or O_RDWR set**.**
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

19 **IF** *PCTS_shm_open* **THEN**

TEST: In the call to *shm_open(name, oflag, mode), any combination of the remaining flags* - O_CREAT, O_EXCL, and O_TRUNC may be specified in *oflag*.
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

20 **IF** *PCTS_shm_open* **THEN**
TEST: The flag O_CREAT has no effect in a call to *shm_open(name, oflag, mode)*that refers to a shared-memory object that was previously created with the O_EXCL flag unset.
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

21 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open(name, oflag, mode)*creates a shared-memory object when the *flag* O_CREAT is set and the shared memory object did not exist previously
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

22 **IF** *PCTS_shm_open* **THEN**
SETUP: Create a shared-memory object by calling *shm_open (name, oflag, mode)* with the *oflag* argument set to O_CREAT
TEST: The user ID of newly created, shared-memory object is set to the effective user ID of the process.
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

23 **IF** *PCTS_shm_open* **THEN**
SETUP: Create a shared-memory object by calling *shm_open (name, oflag, mode)* with the *oflag* argument set to O_CREAT
TEST: The group ID of newly created, shared-memory object is set to the effective user ID of the process.
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

24 **IF** *PCTS_shm_open* **THEN**
TEST: When a shared-memory object is created by calling *shm_open(name, oflag, mode)* with the *oflag* argument set to O_CREAT, the permission bits are set to the value of the *mode* argument, except for those set in the file mode creation mask of the process.
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

D_5 **IF** *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect when bits in *mode* other than the file permission bits are set during creation of a shared-memory object by a call to *shm-open*(*name, oflag, mode*)does so in §12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

25 **IF** *PCTS_shm_open* **THEN**
TEST: In the call *shm_open(name, oflag, mode)* the *mode* argument does not affect whether the shared memory object is opened for reading, for writing, or for both.
TR: Try all three, with all possibly mode bit combinations.
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

26 **IF** *PCTS_shm_open* **THEN**
TEST: A newly created shared memory object has a size of zero
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

R_1 IF *PCTS_shm_open* **THEN**
TEST: When O_EXCL and O_CREAT are set, and the shared memory object already exists, *shm_open*() fails.
ELSE *NO_OPTION*
SEE: Assertion shm_exist_err in §12.3.1.4

27 IF *PCTS_shm_open* **THEN**
TEST: The check by *shm_open()* for the existence of the shared memory object and the creation of the object if it does not exist are atomic, with respect to other processes executing *shm_open*() with O_EXCL and O_CREAT set, and naming the same shared memory object.
NOTE: There is no known reliable test method for this assertion.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_TEST, NO_OPTION

D_6 IF *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of a call to *shm-open*(*name, oflag, mode*) if O_EXCL is set and O_CREAT is not set does so in §12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

28 IF *PCTS_shm_open* **THEN**
TEST: When a shared-memory object exists, a successful call to *shm_open(name, oflag, mode)* specifying both O_RDWR and O_TRUNC, truncates the object to zero length.
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

29 IF *PCTS_shm_open* **THEN**
TEST: When a shared-memory object exists, a successful call to *shm_open(name, oflag, mode)* specifying both O_RDWR and O_TRUNC, leaves the mode and owner unchanged.
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

D_7 IF *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the result of specifying both O_TRUNC and O_RDONLY when calling *shm-open*(*name, oflag, mode*) does so in §12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

30 IF *PCTS_shm_open* **THEN**
TEST: When a shared-memory object is created, by *shm_open( )*, the state of the shared memory object, including all data associated with the shared memory object, persists until the shared memory object is unlinked and all other references are gone.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_8 IF *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the name and state of a shared memory object remain valid after a system reboot does so in §12.3.1.2.
ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

D_9 IF *PCTS_shm_open* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *shm-open*() function does so in §12.3.1.2.

ELSE *NO_OPTION*
Conformance for shm_open: PASS, NO_OPTION

12.3.1.3 Returns

R_2 IF *PCTS_shm_open* **THEN**
TEST: When a call to *shm_open*() completes successfully, the interface returns a nonnegative integer representing the lowest numbered unused file descriptor
ELSE *NO_OPTION*
SEE: Assertion shm_open in §12.3.1.2

R_3 IF *PCTS_shm_open* **THEN**
TEST: When a call to, *shm_open*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error
ELSE *NO_OPTION*
SEE: All assertions in §12.3.1.4

12.3.1.4 Errors

31 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when the shared memory object exists and the permission specified by *oflag* are denied, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

32 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when the shared memory object does not exist and permission to create the shared memory object is denied, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

33 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when O_TRUNC is specified and write permission is denied, returns a value of -1 and sets *errno* to [EACCESS].
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

shm_exist_err
IF *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when O_CREAT and O_EXCL are set and the named shared memory object already exists, returns a value of -1 and sets *errno* to [EEXIST].
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

34 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when the *shm_open*() operation is interrupted by a signal, returns a value of -1 and sets *errno* to [EINTR].
ELSE *NO_OPTION*
Conformance for shm_open: PASS,, NO_OPTION

35 **IF** *PCTS_shm_open* **THEN**
TEST: A call to *shm_open( )*, when the *shm_open*() operation is not supported for the given name, returns a value of -1 and sets *errno* to [EINVAL].

NOTE: A subroutine is recommended that either returns a name for which *shm_open* is not supported, or indicates that there is no way to generate a name for which *shm_open*() is not supported, on the system.

ELSE *NO_OPTION*

Conformance for shm_open: PASS,, NO_OPTION

D_10 **IF** *PCTS_shm_open* **THEN**

TEST: A PCD.1b that documents under what circumstances [EINVALll] may be returned in §12.3.1.2.

ELSE *NO_OPTION*

Conformance for shm_open: PASS, NO_OPTION

36 **IF** *PCTS_shm_open* **THEN**

TEST: A call to *shm_open( )*, when too many file descriptors are currently in use by this process, returns a value of -1 and sets *errno* to [EMFILE].

ELSE *NO_OPTION*

Conformance for shm_open: PASS,, NO_OPTION

37 **IF** *PCTS_shm_open* **THEN**

IF {PATH_MAX} <= *PCTS_PATH_MAX* **THEN**

TEST: A call to *shm_open( )*, when the length of the *name* string exceeds {PATH_MAX} returns a value of -1 and sets *errno* to [ENAMETOOLONG].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for shm_open: PASS, NO_TEST_SUPPORT, NO_OPTION

38 **IF** *PCTS_shm_open* and {_POSIX_NO_TRUNC} **THEN**

IF {NAME_MAX} <= *PCTS_NAME_MAX* **THEN**

TEST: A call to *shm_open( )*, when a pathname component is longer than {NAME_MAX} returns a value of -1 and sets *errno* to [ENAMETOOLONG].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for shm_open: PASS, NO_TEST_SUPPORT, NO_OPTION

39 **IF** *PCTS_shm_open* **THEN**

TEST: A call to *shm_open( )*, when too many shared memory objects are currently open in the system, ,returns a value of -1 and sets *errno* to [EINFILE].

ELSE *NO_OPTION*

Conformance for shm_open: PASS,, NO_OPTION

40 **IF** *PCTS_shm_open* **THEN**

TEST: A call to *shm_open( )*, when O_CREAT is not set and the named shared memory object does not exist, returns a value of -1 and sets *errno* to [ENOENT].

ELSE *NO_OPTION*

Conformance for shm_open: PASS,, NO_OPTION

41 **IF** *PCTS_shm_open* **THEN**

TEST: A call to *shm_open( )*, when there is insufficient space for the creation of the new shared memory object,, returns a value of -1 and sets *errno* to [ENOSPC].

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

ELSE *NO_OPTION*

Conformance for shm_open: PASS,, NO_OPTION

42 **IF** not *PCTS_shm_open* **THEN**

TEST: A call to *shm_open( )*, returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for shm_open: PASS,, NO_OPTION

12.3.2 Remove a Shared Memory Object

Function: *shm_unlink*()

12.3.2.1 Synopsis

1

M_GA_stdC_proto_decl(int; shm_unlink; const char*name,;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for shm_unlink: PASS[1, 2], NO_OPTION

2

M_GA_commonC_result_decl (shm_unlink;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for shm_unlink:: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; shm_unlink;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for shm_open: PASS, NO_OPTION

4

M_GA_macro_args (shm_unlink;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for shm_open: PASS, NO_OPTION

12.3.2.2 Description

shm_unlink

IF *PCTS_shm_unlink* **THEN**
TEST: A successful call to the function *shm_unlink*() removes the name of the shared memory object named by the string pointed to by *name*, and returns zero.
ELSE *NO_OPTION*
Conformance for shm_unlink: PASS,, NO_OPTION

5 **IF** *PCTS_shm_unlink* **THEN**
IF *PCTS_shm_open* **THEN**
TEST: When one or more references to the shared memory object exist when the object is unlinked, the removal of the memory object contents is postponed until all open references to the shared memory object have been removed.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for shm_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** *PCTS_shm_unlink* **THEN**
IF *PCTS_shm_open* and *PCTS_mmap* **THEN**
TEST: When one or more references to the shared memory object exist when the object is unlinked, the removal of the memory object contents is postponed until all map references to the shared memory object have been removed.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for shm_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

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

TEST: A PCD.1b that documents whether or not it supports the *shm_unlink*() function does so in §12.3.2.2.

ELSE *NO_OPTION*

Conformance for shm_unlink: PASS, NO_OPTION

R_1 IF *PCTS_shm_unlink* **THEN**

TEST: When a call to, *shm_unlink*() completes successfully, the interface returns a value of 0.

ELSE *NO_OPTION*

SEE: All assertions in §12.3.2.2

R_2 IF *PCTS_shm_unlink* **THEN**

TEST: When a call to, *shm_unlink*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error, and the named shared memory object is unchanged.

ELSE *NO_OPTION*

SEE: All assertions in §12.3.2.4

12.3.2.4 Errors

7 IF *PCTS_shm_unlink* and {_POSIX_NO_TRUNC} **THEN**

TEST: A call to *shm_unlink( )*, when permission is denied to unlink the named shared memory object, returns a value of -1 and sets *errno* to [EACCESS].

ELSE *NO_OPTION*

Conformance for shm_unlink: PASS, NO_OPTION

8 IF *PCTS_shm_unlink* **THEN**

IF {POSIX_NO_TRUNC} and {NAME_MAX}<= *PCTS_NAME_MAX* **THEN**

TEST: A call to *shm_unlink( )*, when the length of the *name* string exceeds {NAME_MAX} returns a value of -1 and sets *errno* to [ENAMETOOLONG].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for shm_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

9 IF *PCTS_shm_unlink* **THEN**

TEST: A call to *shm_unlink(name)*, when the named shared memory object does not exist, returns a value of -1 and sets *errno* to [ENOENT].

ELSE *NO_OPTION*

Conformance for shm_unlink: PASS, NO_OPTION

10 IF not *PCTS_shm_unlink* **THEN**

TEST: A call to *shm_unlink( )*, returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for shm_unlink: PASS, NO_OPTION

This page is intentionally blank.

Section 13: Execution Scheduling

13.1 Scheduling Parameters

1 **SETUP:** Include the header `<sched.h>`.
TEST: The structure *sched_param* is defined, and has the member

Member Type	Member Name	Description
int	*sched_priority*	Process execution scheduling priority.

Conformance for sched_param: PASS

D_1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents extensions to *sched_param*, as permitted in POSIX.1b{3} §1.3.1.1 item (2), does so in §13.1.
ELSE *NO_OPTION*
Conformance for sched_param: PASS, NO_OPTION

2 **SETUP:** Include the header `<sched.h>`.
TEST: Extensions to *sched_param* that may change the behavior of the application with respect to this standard when those fields in the structure are uninitialized, are enabled as required by POSIX.1b {3} §1.3.1.1.
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.
Conformance for sched_param: PASS, NO_TEST

3 **SETUP:** Include the header `<sched.h>`.
TEST: The symbols allowed by this standard to be in the header `<time.h>` are visible.
Conformance for sched_param: PASS

13.2 Scheduling Policies

1 **TEST:** The implementation makes the process at the head of the highest priority nonempty process list a running process, regardless of its associated policy.
Conformance for sched_policy: *PASS*

2 **TEST:** A running process is then removed from its process list.
NOTE: There is no known portable test method for this assertion.
Conformance for sched_policy: PASS, NO_TEST

D_1 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents scheduling policies other SCHED_FIFO SCHED_RR SCHED_OTHER, does so in §13.2.
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_OPTION

3 **SETUP:** Include the header `<sched.h>`.
TEST: The constants SCHED_FIFO SCHED_RR SCHED_OTHER are defined and are bitwise distinct.
Conformance for sched_policy: PASS

13.2.1 SCHED_FIFO

sched_fifo1
IF *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Processes scheduled under the FIFO scheduling policy are chosen from a process list ordered by the time its processes have been on the list without being executed.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

sched_fifo2
IF *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Under the FIFO scheduling policy, when a running process becomes a preempted process, it becomes the head of the process list for its priority.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

sched_fifo3
IF *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Under the SCHED_FIFO policy, when a blocked process becomes a runnable process, it becomes the tail of the process list for its priority.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

sched_fifo4
IF *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Under the SCHED_FIFO policy, when a running process calls the *sched_setscheduler*() function, the process specified in the function call is modified to the specified policy and the priority specified by the *param* argument.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

sched_fifo5
IF *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Under the SCHED_FIFO policy, if the process whose policy and priority has been modified is a running process or is runnable, it then becomes the tail of the process list for its new priority.

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

sched_fifo6
IF *PCTS_sched_setscheduler* and *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setscheduler* and *PCTS_GAP_sched_setparam* **THEN**
TEST: Under the SCHED_FIFO policy, when a running process calls the *sched_setparam*() function, the priority of the process specified in the function call is modified to the priority specified by the *param* argument.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

sched_fifo7
IF *PCTS_sched_setscheduler* and *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setscheduler* and *PCTS_GAP_sched_setparam* **THEN**
TEST: Under the SCHED_FIFO policy, if a process whose priority has been modified is a running process or is runnable, it then becomes the tail of the process list for its new priority.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

sched_fifo8
IF *PCTS_sched_setscheduler* and *PCTS_sched_yield* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Under the SCHED_FIFO policy, when a running process issues the *sched_yield*() function, the process becomes the tail of the process list for its priority.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

R_1 TEST: Under the SCHED_FIFO policy, the position of a process within the process lists is affected only by process scheduling events.
NOTE: There is no known portable test method for this assertion.
SEE: Assertions sched_fifo1, sched_fifo2, sched_fifo3, sched_fifo4, sched_fifo5, sched_fifo6, sched_fifo7, sched_fifo8 in §13.2.1.

4 **IF** *PCTS_sched_setscheduler* and *PCTS_sched_get_priority_max* and *PCTS_sched_get_priority_min* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: Valid priorities under the SCHED_FIFO policy, are within the range returned by the function *sched_get_priority_max*() and *sched_get_priority_min*() when SCHED_FIFO is provided as the parameter.
TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: The SCHED_FIFO policy has a priority range of at least 32 priorities.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

13.2.2 SCHED_RR

6 **IF** *PCTS_sched_setscheduler* and *PCTS_sched_rr_get_interval* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: The SCHED_RR policy is identical to the SCHED_FIFO policy with the additional condition that when the implementation detects that a running process has been executing as a running process for a time period of the length returned by the function *sched_rr_get_interval*() or longer, the process becomes the tail of its process list and the head of that process list is removed and made a running process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

7 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: A process under the SCHED_RR policy that is preempted and subsequently resumes execution as a running process completes the unexpired portion of its round-robin-interval time period.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_sched_setscheduler* and *PCTS_sched_get_priority_max* and *PCTS_sched_get_priority_min* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: For the SCHED_RR policy, valid priorities are within the range returned by the functions *sched_get_priority_max*() and *sched_get_priority_min*() when SCHED_RR is provided as the parameter.
TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

9 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: The SCHED_RR has a priority range of at least 32 priorities.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

13.2.3 SCHED_OTHER

D_2 TEST: A PCD.1b documents the behavior of the SCHED_OTHER policy as described in the definition of scheduling policy in §13.2.3.
Conformance for sched_policy: PASS

D_3 TEST: The PCD.1b documents the effect of scheduling processes with the SCHED_OTHER policy as described in a system in which other processes are executing under SCHED_FIFO or SCHED_RR, in §13.2.3.
Conformance for sched_policy: PASS

10 **IF** *PCTS_sched_setscheduler* and *PCTS_sched_get_priority_max* and *PCTS_sched_get_priority_min* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: Priorities of processes executing under the SCHED_OTHER policy are restricted to the range returned by the functions *sched_get_priority_max*() and *sched_get_priority_min*() when SCHED_OTHER is provided as the parameter.

TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_policy: PASS, NO_TEST_SUPPORT, NO_OPTION

13.3 Process Scheduling Functions

13.3.1 Set Scheduling Parameters

Function: *sched_setparam*()

13.3.1.1 Synopsis

1

M_GA_stdC_proto_decl(int; sched_setparam; pid_t pid, const struct sched_param *param; sched.h;;;)

SEE: Assertion GA_stdC_proto_decl in §2.7.3

Conformance for sched_setparam: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sched_setparam; sched.h;;;)

SEE: Assertion GA_commonC_int_result_decl in §2.7.3

Conformance for sched_setparam: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sched_setparam; sched.h;;;)

SEE: Assertion GA_macro_result_decl in §1.3.4

Conformance for sched_setparam: PASS, NO_OPTION

4

M_GA_macro_args (sched_setparam; sched.h;;;)

SEE: Assertion GA_macro_args in §2.7.3

Conformance for sched_setparam: PASS, NO_OPTION

13.3.1.2 Description

sched_setparam

IF *PCTS_sched_setparam* **THEN**

IF *PCTS_GAP_sched_setparam* **THEN**

TEST: A successful call to *sched_setparam*() sets the scheduling parameters of the process specified by *pid* to the values specified by the *sched_param* structure pointed to by *param*, and returns the value 0.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_sched_setparam* **THEN**

IF *PCTS_GAP_sched_setparam* **THEN**
TEST: Any integer within the inclusive priority range for the current scheduling policy of the process specified by *pid* is a valid value of the *sched_priority* member in the *param* structure.
TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: Higher numerical values for the priority represent higher priorities.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_sched_setparam* a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of *sched_setparam*() if the value of *pid* is negative, does so in §13.3.1.2.
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_OPTION

7 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: When a process specified by *pid* exists, and if the calling process has permission, the scheduling parameters are set for the process whose process ID is equal to *pid.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

8 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: When *pid* is zero, the scheduling parameters are set for the calling process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_sched_setparam* **THEN**
TEST: A PCD.1b documents the conditions under which one process has permission to change the scheduling parameters of another process, in §13.3.1.2.
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_OPTION

9 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_RAP_sched_setparam* **THEN**
TEST: The requesting process must have the appropriate privilege to set its own scheduling parameters or those of another process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

D_3 IF *PCTS_sched_setparam* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether the requesting process must have appropriate privilege to set its own scheduling parameters or those of another process, does so in §13.3.1.2.
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_OPTION

10 **IF** *PCTS_sched_setparam* THEN
TEST: The target process, whether it is running or not running, resumes execution after all other runnable processes of equal or greater priority have been scheduled to run.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST, NO_OPTION

11 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: When the priority of the process specified by the *pid* argument is set higher than that of the lowest-priority running process and if the specified process is ready to run, the process specified by the *pid* argument preempts a lowest priority running process.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

12 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: When the process calling *sched_setparam*() sets its own priority lower than that of one or more other nonempty process lists, then the process that is the head of the highest priority list preempts the calling process.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

D_4 IF *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparatm* **THEN**
TEST: A PCD.1b documents the result if the current scheduling policy for the process specified by *pid* is not SCHED_FIFO or SCHED_RR, including SCHED_OTHER, in §13.3.1.2.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

D_5 IF *PCTS_sched_setparam* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_setparam*() function does so in §13.3.1.2.
ELSE *NO_OPTION*
Conformance for sched_setparam: PASS, NO_OPTION

13.3.1.4 Errors

13 **IF** *PCTS_sched_setparam* **THEN**
IF *PCTS_GAP_sched_setparam* **THEN**
TEST: A call to *sched_setparam*(), when one or more of the requested scheduling parameters is outside the range defined for the scheduling policy of the specified *pid*, returns a value of -1 and sets *errno* to [EINVAL].

TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

14 **IF** not *PCTS_sched_setparam* **THEN**

TEST: A call to *sched_setparam*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for sched_setparam: PASS, NO_OPTION

15 **IF** *PCTS_sched_setparam* **THEN**

IF *PCTS_GAP_sched_setparam* **THEN**

TEST: A call to *sched_setparam*() when the requesting process does not have permission to set the scheduling parameters for the specified process, returns a value of -1 and sets *errno* to [EPERM].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **IF** *PCTS_sched_setparam* **THEN**

IF *PCTS_RAP_sched_setparam* **THEN**

TEST: A call to *sched_setparam*() when the requesting process does not have the appropriate privilege to invoke *sched_setparam*(), returns a value of -1 and sets *errno* to [EPERM].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

17 **IF** *PCTS_sched_setparam* **THEN**

IF *PCTS_GAP_sched_setparam* **THEN**

TEST: A call to *sched_setparam*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].

NOTE: A subroutine is recommended that returns a *pid* that doesn't correspond to any existing process.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setparam: PASS, NO_TEST_SUPPORT, NO_OPTION

13.3.2 Get Scheduling Parameters

Function: *sched_getparam*()

13.3.2.1 Synopsis

1

M_GA_stdC_proto_decl(int; sched_getparam; pid_t pid, struct sched_param *param; sched.h;;;)

SEE: Assertion GA_stdC_proto_decl in §2.7.3

Conformance for sched_getparam: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sched_getparam; sched.h;;;)

SEE: Assertion GA_commonC_int_result_decl in §2.7.3

Conformance for sched_getparam: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sched_getparam; sched.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sched_getparam: PASS, NO_OPTION

4
M_GA_macro_args (sched_getparam; sched.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sched_getparam: PASS, NO_OPTION

sched_getparam
IF *PCTS_sched_getparam* **THEN**
TEST: A successful call to *sched_getparam*() returns the scheduling parameters of a process specified by *pid* in the *sched_param* structure pointed to by *param*, and returns the value zero.
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

5 **IF** *PCTS_sched_getparam* **THEN**
TEST: When a process specified by *pid* exists and if the calling process has permission, the scheduling parameters for the process whose process ID is equal to *pid* are returned.
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

6 **IF** *PCTS_sched_getparam* **THEN**
TEST: When *pid* is zero, the scheduling parameters for the calling process are returned.
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

D_1 **IF** *PCTS_sched_getparam* and PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of *sched_getparam*() if the value of *pid* is negative, does so in §13.3.2.2.
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

D_2 **IF** *PCTS_sched_getparam* and PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_getparam*() function does so in §13.3.2.2.
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

R_1 **IF** *PCTS_sched_getparam* **THEN**
TEST: When a call to *sched_getparam*() completes successfully, the interface returns zero.
ELSE *NO_OPTION*
SEE: Assertion sched_getparam in §13.3.2.2

R_2 **IF** *PCTS_sched_getparam* **THEN**
TEST: When a call to *sched_getparam*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in in §13.3.2.4

13.3.2.4 Errors

7 **IF** not *PCTS_sched_getparam* **THEN**
TEST: A call to *sched_getparam*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sched_getparam: PASS, NO_OPTION

8 **IF** *PCTS_sched_getparam* **THEN**

TEST: A call to *sched_getparam*(), when the requesting process does not have permission to obtain the scheduling parameters of the specified process, returns a value of -1 and sets *errno* to [EPERM].

ELSE *NO_OPTION*

Conformance for sched_getparam: PASS, NO_OPTION

9 **IF** *PCTS_sched_getparam* **THEN**

TEST: A call to *sched_getparam*(), when No process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].

NOTE: A subroutine is recommended that returns a *pid* that doesn't correspond to any existing process.

ELSE *NO_OPTION*

Conformance for sched_getparam: PASS, NO_OPTION

13.3.3 Set Scheduling Policy and Scheduling Parameters

Function: *sched_setscheduler*()

1

M_GA_stdC_proto_decl(int; sched_setscheduler; pid_t pid, int policy, const struct sched_param *param; sched.h;;;)

SEE: Assertion GA_stdC_proto_decl in §2.7.3

Conformance for sched_setscheduler: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sched_setscheduler; sched.h;;;)

SEE: Assertion GA_commonC_int_result_decl in §2.7.3

Conformance for sched_setscheduler: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sched_setscheduler; sched.h;;;)

SEE: Assertion GA_macro_result_decl in §1.3.4

Conformance for sched_setscheduler: PASS, NO_OPTION

4

M_GA_macro_args (sched_setscheduler; sched.h;;;)

SEE: Assertion GA_macro_args in §2.7.3

Conformance for sched_setscheduler: PASS, NO_OPTION

13.3.3.2 Description

sched_setscheduler

IF *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: A successful call to *sched_setscheduler*() sets the scheduling policy of the process specified by *pid* to *policy*, and its scheduling parameters to and the parameters specified in the *sched_param* structure pointed to by *param*, and returns the former scheduling policy of the specified process.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: The *sched_priority* member in the *param* structure can take on any integer value within the inclusive priority range for the scheduling policy specified by *policy*.

TR: Try to return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

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

TEST: A PCD.1b that documents the behavior of *sched_setscheduler(pid, policy, param*), if the value of *pid* is negative, does so in §13.3.3.2.

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_OPTION

6 IF *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

SETUP: Include the header `<sched.h>`.

TEST: The possible values for the *policy* parameter, in the call *sched_setscheduler (pid, policy, param),* are defined.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

7 IF *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: When a process specified by *pid* exists, and if the calling process has permission, *sched_setscheduler (pid, policy, param)* sets the scheduling policy for the process whose process ID is equal to *pid.*

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

8 IF *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: When a process specified by *pid* exists, and if the calling process has permission, *sched_setscheduler (pid, policy, param)* sets the scheduling parameters for the process whose process is equal to *pid*.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

9 IF *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: When *pid* is zero, *sched_setscheduler (pid, policy, param)* sets the scheduling policy for the calling process.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

10 IF *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: When *pid* is zero, *sched_setscheduler(pid, policy, param)* sets the scheduling parameters for the calling process.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 IF *PCTS_sched_setscheduler* **THEN**

TEST: The PCD.1b documents the conditions under which one process has the appropriate privilege to change the scheduling parameters of another process, in §13.3.3.2.

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_OPTION

D_2 IF *PCTS_sched_setscheduler* THEN

TEST: The PCD.1b documents the conditions under which one process has the appropriate privilege to change the scheduling parameters of another process, in §13.3.3.2.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

D_3 IF *PCTS_sched_setscheduler* and a PCD.1b documents the following **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: A PCD.1 that documents whether the requesting process has permission to set its own scheduling parameters or those of another process, does so in §13.3.3.2.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

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

TEST: A PCD.1 that documents restrictions that apply as to the appropriate privileges required to set a process's own scheduling policy, or another process's scheduling policy, to a particular value, does so in §13.3.3.2.

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_OPTION

R_1 IF *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: The *sched_setscheduler*() function is considered successful if it succeeds in setting the scheduling policy and scheduling parameters of the process specified by *pid* to the values specified by *policy* and the structure *param*, respectively.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion sched_setscheduler in §13.3.3.2

D_5 IF *PCTS_sched_setscheduler* and a PCD.1b documents the following **THEN**

TEST: A PCD.1 that documents whether or not it supports the *sched_setscheduler*() function, does so in §13.3.3.2.

ELSE *NO_OPTION*

Conformance for sched_setscheduler: PASS, NO_OPTION

13.3.3.3 Returns

R_2 IF *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: When a call to *sched_setscheduler*() completes successfully, the interface returns the former scheduling policy of the specified process.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

SEE: Assertion sched_setscheduler in §13.3.3.2

R_3 IF *PCTS_sched_setscheduler* **THEN**

IF *PCTS_GAP_sched_setscheduler* **THEN**

TEST: When a call to *sched_setscheduler*() completes unsuccessfully, the policy and scheduling parameters remain unchanged, and the interface returns a value of -1 and sets *errno* to indicate the error.

ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: All assertions in §13.3.3.4

13.3.3.4 Errors

11 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: A call to *sched_setscheduler*(), when the value of the *policy* parameter is invalid, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either invalid value for *policy* or indicates that there is no way to generate invalid value for *policy* on the system.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

12 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: A call to *sched_setscheduler*(), when one or more of the parameters contained in *param* is outside the valid range for the specified scheduling policy, returns a value of -1 and sets *errno* to [EINVAL].
TR: Try the return values of *sched_get_priority_max*() and *sched_get_priority_min*(). If *sched_get_priority_max*() returns a value less than INT_MAX, try *sched_get_priority_max*()+1. If *sched_get_priority_min*() returns a value greater than INT_MIN, try *sched_get_priority_min*()-1.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_OPTION

13 **IF** not *PCTS_sched_setscheduler* **THEN**
TEST: A call to *sched_setscheduler*(), returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_OPTION

14 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: A call to *sched_setscheduler*(), when the requesting process does not have permission to set the scheduling parameters of the specified process, returns a value of -1 and sets *errno* to [EPERM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: A call to *sched_setscheduler*(), when the requesting process does not have permission to set the scheduling policy of the specified process, returns a value of -1 and sets *errno* to [EPERM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for sched_setscheduler: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **IF** *PCTS_sched_setscheduler* **THEN**
IF *PCTS_GAP_sched_setscheduler* **THEN**
TEST: A call to *sched_setscheduler*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].
ELSE *NO_TEST_SUPPORT*

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

13.3.4 Get Scheduling Policy

Function: *sched_getscheduler*()

13.3.4.1 Synopsis

1
M_GA_stdC_proto_decl(int; sched_getscheduler; pid_t pid; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sched_getscheduler: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(sched_getscheduler; sched.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sched_getscheduler: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; sched_getscheduler; sched.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sched_getscheduler: PASS, NO_OPTION

4
M_GA_macro_args (sched_getscheduler; sched.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sched_getscheduler: PASS, NO_OPTION

13.3.4.2 Description

sched_getscheduler
IF *PCTS_sched_getscheduler* **THEN**
TEST: A successful call to *sched_getscheduler*() returns the scheduling policy of the process specified by *pid*.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

5 **IF** *PCTS_sched_getscheduler* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the behavior of *sched_getscheduler*() if the value of *pid* is negative, does so in §13.3.4.2.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

6 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: The values that can be returned by *sched_getscheduler*() are defined in the header file `<sched.h>`.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

7 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: When a process specified by *pid* exists, and if the calling process has permission, the scheduling policy is returned for the process whose process ID is equal to *pid*.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

8 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: When *pid* is zero, the scheduling policy is returned for the calling process.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

D_1 **IF** *PCTS_sched_getscheduler* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_getscheduler*() function does so in §13.3.4.2.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

13.3.4.3 Returns

R_1 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: When a call to *sched_getscheduler*() completes successfully, the interface returns the former policy of the specified process.
ELSE *NO_OPTION*
SEE: Assertion sched_getscheduler in §13.3.4.2

R_2 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: When a call to *sched_getscheduler*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §13.3.4.4

13.3.4.4 Errors

9 **IF** not *PCTS_sched_getscheduler* **THEN**
TEST: A call to *sched_getscheduler*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

10 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: A call to *sched_getscheduler*() when the requesting process does not have permission to determine the scheduling policy of the specified process, returns a value of -1 and sets *errno* to [EPERM].
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

11 **IF** *PCTS_sched_getscheduler* **THEN**
TEST: A call to *sched_getscheduler*() when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH].
NOTE: A subroutine is recommended that returns a *pid* that doesn't correspond to any existing process.
ELSE *NO_OPTION*
Conformance for sched_getscheduler: PASS, NO_OPTION

13.3.5 Yield Processor

Function: *sched_yield*()

13.3.5.1 Synopsis

1
M_GA_stdC_proto_decl(int; sched_yield; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sched_yield: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sched_yield; sched.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sched_yield: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sched_yield; sched.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sched_yield: PASS, NO_OPTION

4

M_GA_macro_args (sched_yield; sched.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sched_yield: PASS, NO_OPTION

13.3.5.2 Description

sched_yield

IF *PCTS_sched_yield* **THEN**
TEST: A successful call to *,;9sched_yield*() forces the running process to relinquish the process until it again becomes the head of its process list, and returns the value zero.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sched_yield: PASS, NO_TEST, NO_OPTION

D_1 IF *PCTS_sched_yield* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_yield*() function does so in §13.3.5.2.
ELSE *NO_OPTION*
Conformance for sched_yield: PASS, NO_OPTION

13.3.5.3 Returns

R_1 IF *PCTS_sched_yield* **THEN**
TEST: When a call to *sched_yield*() completes successfully, the interface returns 0.
ELSE *NO_OPTION*
SEE: Assertion sched_yield in §13.3.5.2

R_2 IF *PCTS_sched_yield* **THEN**
TEST: When a call to *sched_yield*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §13.3.5.4

13.3.5.4 Errors

5 **IF** not *PCTS_sched_yield* **THEN**
TEST: A call to *sched_yield*() returns a value of -1 and sets *errno* to [ENOSYS]
ELSE *NO_OPTION*
Conformance for sched_yield: PASS, NO_OPTION

13.3.6 Get Scheduling Parameter Limits

Functions:
sched_get_priority_max(),

sched_get_priority_min(),
sched_rr_get_interval()

13.3.6.1 Synopsis

1

M_GA_stdC_proto_decl(int; sched_get_priority_max; int policy; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sched_get_priority_max: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(sched_get_priority_max; sched.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sched_get_priority_max: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; sched_get_priority_max; sched.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sched_get_priority_max: PASS, NO_OPTION

4

M_GA_macro_args (sched_get_priority_max; sched.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sched_get_priority_max: PASS, NO_OPTION

5

M_GA_stdC_proto_decl(int; sched_get_priority_min; , int policy; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sched_get_priority_min: PASS[5,6], NO_OPTION

6

M_GA_commonC_int_result_decl(sched_get_priority_min; , int policy; sched.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sched_get_priority_min: PASS[5, 6], NO_OPTION

7

M_GA_macro_result_decl(int; sched_get_priority_min; sched.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sched_get_priority_min: PASS, NO_OPTION

8

M_GA_macro_args (sched_get_priority_min; sched.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sched_get_priority_min: PASS, NO_OPTION

9

M_GA_stdC_proto_decl(int; sched_rr_get_interval; , pid_t pid, struct timespec *interval; sched.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for sched_rr_get_interval: PASS[9, 10], NO_OPTION

10

M_GA_commonC_int_result_decl(sched_rr_get_interval; , pid_t pid, struct timespec *interval; sched.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for sched_rr_get_interval: PASS[9, 10], NO_OPTION

11

M_GA_macro_result_decl(int; sched_rr_get_interval; sched.h;;;)

SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for sched_rr_get_interval: PASS, NO_OPTION

12
M_GA_macro_args (sched_rr_get_interval; sched.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for sched_rr_get_interval: PASS, NO_OPTION

13.3.6.2 Description

sched_get_priority_max
IF *PCTS_sched_get_priority_max* **THEN**
TEST: A successful call to *sched_get_priority_max*() returns the appropriate maximum for the scheduling policy specified by *policy*.
TR: Test for SCHED_FIFO, SCHED_RR, and SCHED_OTHER
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

sched_get_priority_min
IF *PCTS_sched_get_priority_min* **THEN**
TEST: A successful call to *sched_get_priority_min* () returns the appropriate minimum for the scheduling policy specified by *policy*.
TR: Test for SCHED_FIFO, SCHED_RR, and SCHED_OTHER
ELSE *NO_OPTION*
Conformance for sched_get_priority_min: PASS, NO_OPTION

sched_rr_get_interval
IF *PCTS_sched_rr_get_interval* **THEN**
TEST: A successful call to *sched_rr_get_interval*() updates the *timespec* structure referenced by the *interval* argument to contain the current execution time limit (i.e., time quantum) for the process specified by *pid*, and returns the value zero.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sched_rr_get_interval: PASS, NO_TEST, NO_OPTION

13 **IF** *PCTS_sched_rr_ get_interval* **THEN**
TEST: When *pid* is zero, *sched_rr_get_interval*() returns the current execution time limit for the calling process.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for sched_rr_get_interval: PASS, NO_TEST, NO_OPTION

14 **IF** *PCTS_sched_get_priority_max* **THEN**
TEST: In the call to *sched_get_priority_max(policy)*, the value of *policy* is one of the scheduling policy values defined in `<sched.h>`.
NOTE: A subroutine is recommended that either returns an invalid scheduling policy or indicates that there is no way to generate an invalid scheduling policy on the system.
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

15 **IF** *PCTS_sched_get_priority_min* **THEN**
TEST: In the call *sched_get_priority_min(policy)*, the value of *policy* is one of the scheduling policy values defined in `<sched.h>`.
NOTE: A subroutine is recommended that either returns an invalid scheduling policy or indicates that there is no way to generate an invalid scheduling policy on the system.
ELSE *NO_OPTION*
Conformance for sched_get_priority_min: PASS, NO_OPTION

D_1 IF *PCTS_sched_get_priority_max* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_get_priority_max*() function does so in §13.3.6.2.
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

D_2 IF *PCTS_sched_get_priority_min* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_get_priority_min*() function does so in §13.3.6.2.
ELSE *NO_OPTION*
Conformance for sched_get_priority_min: PASS, NO_OPTION

D_3 IF *PCTS_sched_rr_get_interval* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *sched_rr_get_interval*() function does so in §13.3.6.2.
ELSE *NO_OPTION*
Conformance for sched_rr_get_interval: PASS, NO_OPTION

13.3.6.3 Returns

R_1 IF *PCTS_sched_get_priority_max* **THEN**
TEST: When a call to *sched_get_priority_max*() completes successfully, the interface returns the appropriate maximum value.
ELSE *NO_OPTION*
SEE: Assertion sched_get_priority_max in §13.3.6.2

R_2 IF *PCTS_sched_get_priority_min* **THEN**
TEST: When a call to *sched_get_priority_min*() completes successfully, the interface returns the appropriate minimum value.
ELSE *NO_OPTION*
SEE: Assertion sched_get_priority_min in §13.3.6.2

R_3 IF *PCTS_sched_get_priority_max* **THEN**
TEST: When a call to *sched_get_priority_max*() completes successfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §13.3.6.4

R_4 IF *PCTS_sched_get_priority_min* **THEN**
TEST: When a call to *sched_get_priority_min*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §13.3.6.4

R_5 IF *PCTS_sched_rr_get_interval* **THEN**
TEST: When a call to *sched_rr_get_interval*(, completes successfully, the interface returns zero.
ELSE *NO_OPTION*
SEE: Assertion sched_rr_get_interval in §13.3.6.2

R_6 IF *PCTS_sched_rr_get_interval* **THEN**
TEST: When a call to *sched_rr_get_interval*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §13.3.6.4

13.3.6.4 Errors

14 **IF** *PCTS_sched_get_priority_max* **THEN**
TEST: A call to *sched_get_priority_max*() when the value of the *policy* parameter does not represent a defined scheduling policy, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid scheduling policy or indicates that there is no way to generate an invalid scheduling policy on the system.
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

17 **IF** *PCTS_sched_get_priority_min* **THEN**
TEST: A call to *sched_get_priority_min*() when the value of the *policy* parameter does not represent a defined scheduling policy, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid scheduling policy or indicates that there is no way to generate an invalid scheduling policy on the system.
ELSE *NO_OPTION*
Conformance for sched_get_priority_min: PASS, NO_OPTION

18 **IF** not *PCTS_sched_get_priority_max* **THEN**
TEST: A call to *sched_get_priority_max*() returns a value of -1 and sets *errno* to [ENOSYS]
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

19 **IF** not *PCTS_sched_get_priority_min* **THEN**
TEST: A call to *sched_get_priority_min*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sched_get_priority_min: PASS, NO_OPTION

20 **IF** not *PCTS_sched_rr_get_interval* **THEN**
TEST: A call to *sched_rr_get_interval*()returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for sched_rr_get_interval: PASS, NO_OPTION

21 **IF** *PCTS_sched_get_priority_max* **THEN**
TEST: A call to *sched_get_priority_max*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH]
ELSE *NO_OPTION*
Conformance for sched_get_priority_max: PASS, NO_OPTION

22 **IF** *PCTS_sched_get_priority_min* **THEN**
TEST: A call to *sched_get_priority_min*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH.]
NOTE: A subroutine is recommended that returns a *pid* that doesn't correspond to any existing process.
ELSE *NO_OPTION*
Conformance for sched_get_priority_min: PASS, NO_OPTION

23 **IF** *PCTS_sched_rr_get_interval* **THEN**
TEST: A call to *sched_rr_get_interval*(), when no process can be found corresponding to that specified by *pid*, returns a value of -1 and sets *errno* to [ESRCH]
NOTE: A subroutine is recommended that returns a *pid* that doesn't correspond to any existing process.
ELSE *NO_OPTION*
Conformance for sched_rr_get_interval: PASS, NO_OPTION

Section 14: Clocks and Timers

14.1 Data Definitions for Clocks and Timers

14.1.1 Time Value Specification Structures

1 **SETUP:** Include the header `<time.h>`.
TEST: The structure *timespec* is defined, and includes the members

Member Type	Member Name	Description
time_t	*tv_sec*	Seconds
long	*tv_nsec*	Nanoseconds

Conformance for timer_hdr: PASS

D_1 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents extensions to *timespec*, as permitted in POSIX.1b{3} §1.3.1.1 item (2), does so in §14.1.1.
ELSE *NO_OPTION*
Conformance for timer_hdr: PASS, NO_OPTION

2 **SETUP:** Include the header `<time.h>`.
TEST: Extensions to *timespec* that may change the behavior of the application with respect to this standard when those fields in the structure are uninitialized, are enabled as required by POSIX.1b {3} §1.3.1.1.
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.
Conformance for timer_hdr: PASS, NO_TEST

3 **TEST:** The *tv_nsec* member of *timespec* is only valid if less than the number of nanoseconds in a second (1000 million).
Conformance for timer_hdr: PASS

4 **IF** *{int_max}* <=10E9 **THEN**
TEST: The *tv_nsec* member of *timespec* is only valid if less than the number of nanoseconds in a second (1000 million).
ELSE *NO_TEST_SUPPORT*
Conformance for timer_hdr: PASS, NO_TEST_SUPPORT

5 **TEST:** The time interval described by *timespec is (tv_sec* x 10^9 + *tv_nsec*) nanoseconds.
Conformance for timer_hdr: PASS

6 **SETUP:** Include the header `<time.h>`.
TEST: The structure *itimerspec* is defined, and includes the members

Member Type	Member Name	Description
struct timespec	*it_interval*	Timer period
struct timespec	*it_value*	Timer expiration

Conformance for timer_hdr: PASS

D_2 IF a PCD.1b documents the following **THEN**

TEST: A PCD.1b documents extensions to *itimerspec*, as permitted in POSIX1b{3} §1.3.1.1 item (2), does so in §14.1

Conformance for timer_hdr:PASS, NO_OPTION

7 **SETUP:** Include the header `<time.h>`.

TEST: Extensions to *itimerspec* that may change the behavior of the application with respect to this standard when those fields in the structure are uninitialized, are enabled as required by POSIX.1b {3} §1.3.1.1.

NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.

Conformance for timer_hdr: PASS, NO_TEST

8 **TEST:** When the value described by *it_value* is nonzero, it indicates the time to or time of the next timer expiration (for relative and absolute timer values, respectively).

Conformance for timer_hdr: PASS

9 **TEST:** When the value described by *it_value* is zero, the timer is disarmed..

Conformance for timer_hdr: PASS

10 **TEST:** When the value described by *it_interval* is nonzero, it specifies an interval to be used in reloading the timer when it expires – that is, a periodic timer is specified.

Conformance for timer_hdr: PASS

11 **TEST:** When the value described by *it_interval* is zero, the timer is disarmed after its next expiration – that is, a "one-shot" timer is specified.

Conformance for timer_hdr: PASS

14.1.2 Timer Event Notification Control Block

12 **IF** {_POSIX_REALTIME_SIGNALS}**THEN**

TEST: Per-process timers can be created that notify the process of timer expirations by queuing a realtime extended signal.

ELSE *NO_TEST_SUPPORT*

Conformance for timer_hdr: PASS, NO_TEST_SUPPORT

14.1.3 Type Definitions

13 **SETUP:** Include the header `<sys/types.h>`.

TEST: The types *clockid_t* and *timer_t* are defined.

Conformance for timer_hdr: PASS

14.1.4 Manifest Constants

14 **SETUP:** Include the header `<time.h>`.

TEST: The constants CLOCK_REALTIME and TIMER_ABSTIME are defined.

Conformance for timer_hdr: PASS

15 **SETUP:** Include the header <unistd.h>.
TEST: The constant {_POSIX_CLOCKRES_MIN} and is defined as 20 ms (1/50 of a second).
Conformance for timer_hdr: PASS

16 **TEST:** The maximum allowable resolution for the CLOCK_REALTIME clock and all times based on this clock, including the *nanosleep*() function, is {_POSIX_CLOCKRES_MIN}.
Conformance for timer_hdr: PASS

D_3 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents support for smaller values of resolution for the CLOCK_REALTIME clock to provide finer granularity time bases, does so in §14.1.4.
ELSE *NO_OPTION*
Conformance for timer_hdr: PASS, NO_OPTION

17 **TEST:** The actual resolution for a specific clock is obtained using functions defined in this chapter.
Conformance for timer_hdr: PASS

D_4 **IF** a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents does so in §14.1.4.
ELSE *NO_OPTION*
Conformance for timer_hdr: PASS, NO_OPTION

D_5 **TEST:** The PCD.1b documents the actual resolution supported for the *nanosleep*() function of timers based on this clock, if it differs from the resolution supported for the clock, in §14.1.4.
Conformance for timer_hdr: PASS

18 **TEST:** The maximum value of CLOCK_RALTIME clock and absolute timers based on it is at least that defined by the C Standard {2{ for the *time_t* type.
Conformance for timer_hdr: PASS

D_6 **TEST:** The PCD.1b documents the difference between the maximum value supported by the *nanosleep*() function or timers based on this clock and the maximum value supported by the clock, if such a difference exists, in §14.1.4.
Conformance for timer_hdr: PASS

14.2 Clocks and Timer Functions

14.2.1 Clocks

Functions: *clock_settime*(), *clock_gettime*(), *clock_getres*()

14.2.1.1 Synopsis

1
M_GA_stdC_proto_decl(int; clock_settime; clockid_t clock_id, const struct timespec *tp; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for clock_settime: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(clock_settime; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for clock_settime: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; clock_settime; time.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for clock_settime: PASS, NO_OPTION

4

M_GA_macro_args (clock_settime; time.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for clock_settime: PASS, NO_OPTION

5

M_GA_stdC_proto_decl(int; clock_gettime; clockid_t clock_id, struct timespec *tp; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for clock_gettime: PASS[5, 6], NO_OPTION

6

M_GA_commonC_int_result_decl(clock_gettime; , clockid_t, clock_id, struct timespec *tp; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for clock_gettime: PASS[5, 6], NO_OPTION

7

M_GA_macro_result_decl(int; clock_gettime; time.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for clock_gettime: PASS, NO_OPTION

8

M_GA_macro_args (clock_gettime; time.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for clock_gettime: PASS, NO_OPTION

9

M_GA_stdC_proto_decl(int; clock_getres; , clockid_t clock_id, struct timespec *res; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for clock_getres: PASS[9, 10], NO_OPTION

10

M_GA_commonC_int_result_decl(clock_getres; , clockid_t clock_id, struct timespec *res; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for clock_getres: PASS[9, 10], NO_OPTION

11

M_GA_macro_result_decl(int; clock_getres; time.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for clock_getres: PASS, NO_OPTION

12

M_GA_macro_args (clock_getres; time.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for clock_getres: PASS, NO_OPTION

1

M_GA_stdC_proto_decl(int; clock_settime; clockid_t clock_id, const struct timespec *tp; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for clock_settime: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(clock_settime; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for clock_settime: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; clock_settime; time.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for clock_settime: PASS, NO_OPTION

4

M_GA_macro_args (clock_settime; time.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for clock_settime: PASS, NO_OPTION

14.2.1.2 Description

clock_settime
IF *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: A successful call to *clock_settime (clock_id, tp)* sets the specified clock, *clock_id*, to the value specified by *tp*, and returns zero.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

13 **IF** *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: In a call to *clock_settime*(), time values that are between two consecutive non-negative integer multiples of the resolution of the specified clock are truncated down to the smaller multiple of the resolution.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

clock_gettime
IF *PCTS_clock_gettime* **THEN**
TEST: A successful call to *clock_gettime (clock_id, tp)* returns the current value *tp* for the specified clock, *clock_id*.
ELSE *NO_OPTION*
Conformance for clock_gettime: PASS, NO_OPTION

clock_getres
IF *PCTS_clock_getres* **THEN**
TEST: A successful call to *clock_getres (clock_id, res)*, with a non_**NULL** value of the argument *res*, stores the resolution of the clock specified by *clock_id* into the location pointed to by *res*, and returns zero.
ELSE *NO_OPTION*
Conformance for clock_getres: PASS, NO_OPTION

14 **IF** *PCTS_clock_getres* **THEN**
TEST: The PCD.1b documents clock resolutions, in §14.2.1.2.
ELSE *NO_OPTION*
Conformance for clock_getres: PASS, NO_OPTION

15 **TEST:** Clock resolutions cannot be set by a process.
Conformance for clock_settime: PASS

16 **IF** *PCTS_clock_getres* **THEN**
TEST: When the *res* argument to the call *clock_getres (, clock_id, res)* is **NULL**, the clock resolution is not returned.
ELSE *NO_OPTION*
Conformance for clock_getres: PASS, NO_OPTION

17 **IF** *PCTS_clock_settime* **THEN**
IF *PCTS__clock_getres* and *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**

TEST: When the time argument of *clock_settime*() is not a multiple of the clock resolution, *res*, as determined by a call to *clock_getres(, clock_id, res)*, then the value is truncated to a multiple of *res*.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 TEST: The PCD.1b documents whether the clocks are systemwide – that is, visible to all processes; or per-process – measuring time that is meaningful only within a process. in §14.2.1.2.

Conformance for clock_settime: PASS

18 **SETUP:** Include the header `<time.h>`.

TEST: The constant CLOCK_REALTIME is defined.

Conformance for clock_settime: PASS

19 **TEST:** When the value CLOCK_REALTIME is used for *clock_id*, the call *clock_settime(clock_id, tp)* sets the realtime clock for the system.

Conformance for clock_settime: PASS

20 **TEST:** When the value CLOCK_REALTIME is used for *clock_id*, the call *clock_gettime(clock_id, tp)* interrogates the realtime clock for the system.

Conformance for clock_settime: PASS

21 **TEST:** When the value CLOCK_REALTIME is used for *clock_id*, the call *clock_get_res (, clock_id, res)* gets the resolution of the realtime clock for the system.

Conformance for clock_settime: PASS

22 **IF** *PCTS_clock_gettime* **THEN**

TEST: When the value CLOCK_REALTIME is used for *clock_id*, the value returned by *clock_gettime(clock_id, tp)* represents the amount of time (in seconds and nanoseconds) since the Epoch.

ELSE *NO_OPTION*

Conformance for clock_gettime: PASS, NO_OPTION

23 **IF** *PCTS_clock_settime* **THEN**

IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**

TEST: When the value CLOCK_REALTIME is used for *clock_id*, the value specified by *clock_settime(clock_id, tp)* represents the amount of time (in seconds and nanoseconds) since the Epoch.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

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

TEST: A PCD.1b that documents whether clocks in addition to the realtime clock are supported, and the interpretation of time values for any such clocks, does so in §14.2.1.2.

ELSE *NO_OPTION*

Conformance for clock_settime: PASS, NO_OPTION

24 **IF** *PCTS_clock_settime* **THEN**

TEST: The PCD.1b documents the effect of setting a clock via *clock_settime*() on armed per-process times associated with that clock, in §14.2.1.2.

ELSE *NO_OPTION*

Conformance for clock_settime: PASS, NO_OPTION

25 **IF** *PCTS_clock_settime* **THEN**

TEST: The PCD.1b documents the appropriate privilege to set a particular clock, in §14.2.1.2.

ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_OPTION

D_3 IF *PCTS_clock_settime* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *clock_settime*() function does so in §14.2.1.2.
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_OPTION

D_4 IF *PCTS_clock_gettime* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *clock_gettime*() function does so in §14.2.1.2.
ELSE *NO_OPTION*
Conformance for clock_gettime: PASS, NO_OPTION

D_5 IF *PCTS_clock_getres* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *clock_getres*() function does so in §14.2.1.2.
ELSE *NO_OPTION*
Conformance for clock_getres: PASS, NO_OPTION

14.2.1.3 Returns

R_1 IF *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: When a call to *clock_settime*() completes successfully, the interface returns a value of 0.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: Assertion clock_settime in §14.2.1.4

R_2 IF *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: When a call to *clock_settime*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
clock_settime()
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
SEE: All assertions in §14.2.1.4 controlled by *clock_settime*()

R_3 IF *PCTS_clock_gettime* THEN
TEST: When a call to *clock_gettime*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion clock_gettime in §14.2.1.4

R_4 IF *PCTS_clock_gettime* THEN
TEST: When a call to *clock_gettime*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
clock_gettime()
ELSE *NO_OPTION*
SEE: All assertions in §14.2.1.4 controlled by *clock_gettime*()

R_5 IF *PCTS_clock_getres* **THEN**
TEST: When a call to *clock_getres*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion clock_getres in §14.2.1.4

R_6 IF *PCTS_clock_getres* **THEN**
TEST: When a call to *clock_gettime*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §14.2.1.4 controlled by *clock_getres*()

14.2.1.4 Errors

26 **IF** *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: A call to *clock_settime (clock_id, tp)* when the *clock_id* argument does not specify a known clock, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid clock name or indicates that there is no way to generate an invalid clock name on the system.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

27 **IF** *PCTS_clock_gettime* THEN
TEST: A call to *clock_gettime (, clock_id, tp)* when the *clock_id* argument does not specify a known clock, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid clock name or indicates that there is no way to generate an invalid clock name on the system.
ELSE *NO_OPTION*
Conformance for clock_gettime: PASS, NO_OPTION

28 **IF** *PCTS_clock_getres* **THEN**
TEST: A call to *clock_getres (, clock_id, res)* when the *clock_id* argument does not specify a known clock, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid clock name or indicates that there is no way to generate an invalid clock name on the system.
ELSE *NO_OPTION*
Conformance for clock_getres: PASS, NO_OPTION

29 **IF** not *PCTS_clock_gettime* **THEN**
TEST: A call to *clock_settime*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_OPTION

30 **IF** not *PCTS_clock_gettime* **THEN**
TEST: A call to *clock_gettime*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for clock_gettime: PASS, NO_OPTION

31 **IF** not *PCTS_clock_getres* **THEN**
TEST: A call to *clock_getres*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for clock_getres: PASS, NO_OPTION

32 **IF** *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: A call to *clock_settime (clock_id, tp)* when the *tp* argument is outside the range for te given clock id, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns a value outside the range value for any given clock id or indicates that there is no way to generate a value outside the range value for any given clock id on the system.
ELSE *NO_TEST_SUPPORT*

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

33 **IF** *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: A call to *clock_settime (clock_id, tp)* when the *tp* argument specified a nanosecond value less than zero, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

34 **IF** *PCTS_clock_settime* **THEN**
IF *PCTS_GAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* and {INT_MAX} <=10e9 **THEN**
TEST: A call to *clock_settime (clock_id, tp)* when the *tp* argument specified a nanosecond value greater than or equal to 1000 million, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

35 **IF** *PCTS_clock_settime* **THEN**
IF *PCTS_RAP_clock_settime* and *PCTS_DETECT_EPERM_clock_settime* **THEN**
TEST: A call to *clock_settime*() when the requesting process does not have the appropriate privilege to set the specified clock, returns a value of -1 and sets *errno* to [EPERM].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for clock_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

14.2.2 Create a Per-Process Timer

Function: *timer_create*()

14.2.2.1 Synopsis

1
M_GA_stdC_proto_decl(int; timer_create; clockid_t clock_id, struct sigevent *evp, timer_t *timerid; signal.h; time.h;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for timer_create: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(timer_create; signal.h; time.h;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for timer_create: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; timer_create; signal.h; time.h;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for timer_create: PASS, NO_OPTION

4
M_GA_macro_args (timer_create; signal.h; time.h;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for timer_create: PASS, NO_OPTION

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 §14.2.2.4), 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**

TEST: The PCD.1b documents the behavior for any value of *sigev_notify*, other than SIGEV_NONE or SIGEV_SIGNAL in §14.2.2.2.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

14 **IF** *PCTS_timer_create* **THEN**

TEST: There is a set of clocks that can be used as timing bases for per-process timers.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

15 **IF** *PCTS_timer_create* **THEN**

TEST: There is at least one mechanism for notifying the process of timer expiration events.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

D_1 **IF** *PCTS_timer_create* **THEN**

TEST: The PCD.1b documents the set of clocks that can be used as timing bases for per-process timers and one or more mechanisms for notifying the process of timer expiration events, in §14.2.2.2.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

16 **IF** *PCTS_timer_create* **THEN**

SETUP: Include the header `<time.h>`.

TEST: The constant CLOCK_REALTIME is defined.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

17 **IF** *PCTS_timer_create* **THEN**

TEST: In a call to *timer_create(clock_id, evp, timerid)*, CLOCK_REALTIME is a legitimate value for *clock_id*.

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_OPTION

18 **IF** *PCTS_timer_create* **THEN**

IF {_POSIX_REALTIME_SIGNALS} **THEN**

SETUP: Call *timer_create(clock_id, evp, timerid)*, with *evp* argument pointing to a *sigevent* structure that specifies SIGEV_SIGNAL, and SA_SIGINFO set for the expiration signal.

TEST: The signal and application-defined value specified in the *sigevent* structure are queued to the process on timer expiration.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_timer_create* **THEN**

IF {_POSIX_REALTIME_SIGNALS} **THEN**

SETUP: Call *timer_create(clock_id, evp, timerid)*, with *evp* argument set to **NULL** and SA_SIGINFO set for the expiration signal.

TEST: A default signal is queued to the process and the signal data value is set to the timer ID.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for timer_create: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_timer_create* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether the realtime signal is queued and what value, if any, is sent, if SA_SIGINFO is not set for the expiration signal, does so in §14.2.2.2.

ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

20 **IF** *PCTS_timer_create* **THEN**
IF not {_POSIX_REALTIME_SIGNALS} **THEN**
SETUP: Call *timer_create(clock_id, evp, timerid),* with *evp* pointing to a *sigevent* structure that specifies SIGEV_SIGNAL.
TEST: The signal number defined in the *sigevent* structure is sent to the process on timer expiration.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **IF** *PCTS_timer_create* **THEN**
IF not {_POSIX_REALTIME_SIGNALS} **THEN**
TEST: Following the call *timer_create(clock_id, evp, timerid),* if *evp* is **NULL**, then a default signal is sent to the process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_TEST_SUPPORT, NO_OPTION

22 **IF** *PCTS_timer_create* **THEN**
SETUP: Call *timer_create(clock_id, evp, timerid),* with a *clock_id* value of CLOCK_REALTIME, and an *evp* of **NULL.**
TEST: The default signal is SIGALRM.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

23 **IF** *PCTS_timer_create* **THEN**
TEST: The PCD.1b documents the default signal number for any clock other than CLOCK_REALTIME, in §14.2.2.2.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

24 **IF** *PCTS_timer_create* **THEN**
TEST: Per-process timers are not inherited by a child process across a *fork*().
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

25 **FOR:** *execl*(), *execv*(), *execle*(), *execve*(), *exelp*(), and *execvp*()
IF *PCTS_timer_create* **THEN**
TEST: Per-process timers are disarmed and deleted by a successful call to *function*().
NOTE: The assertion is tested once for each function specified in the FOR clause. The assertion is to be read by substituting *function*() with the current function specified in the FOR clause. The name of the function also is to be substituted for each occurrence in the construct *PCTS_function.*
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

D_3 **IF** *PCTS_timer_create* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *timer_create*() function does so in §14.2.2.2.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

14.2.2.3 Returns

R_1 IF *PCTS_timer_create* **THEN**
TEST: When a call to *timer_create(clock_id, evp, timerid)* completes successfully, the interface returns a value of 0, and updates the location referenced by *timerid* to a *timer_t*, which can be passed to the per-process timer calls (see §14.2.4).
ELSE *NO_OPTION*
SEE: Assertion timer_create in §14.2.2.4

R_2 IF *PCTS_timer_create* **THEN**
TEST: When a call to *timer_create(clock_id, evp, timerid)* completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §14.2.2.4

D_4 IF *PCTS_timer_create* and A PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the value of *timerid* if an error occurs, following a call to *timer_create(clock_id, evp, timerid*), does so in §14.2.2.4.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

14.2.2.4 Errors

26 **IF** *PCTS_timer_create* **THEN**
TEST: A call to *timer_create*(), when the system lacks sufficient signal queuing resources to honor the request, returns a value of -1 and sets *errno* to [EAGAIN].
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

27 **IF** *PCTS_timer_create* **THEN**
IF {TIMER_MAX} <= *PCTS_TIMER_MAX* **THEN**
TEST: A call to *timer_create*(), when the calling process has already created all of the timers it is allowed by this implementation, returns a value of -1 and sets *errno* to [EAGAIN].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_TEST_SUPPORT, NO_OPTION

28 **IF** *PCTS_timer_create* **THEN**
TEST: A call to *timer_create*(), when the specified clock ID is not defined, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns a value outside the range value for any given clock id or indicates that there is no way to generate a value outside the range value for any given clock id on the system.
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

29 **IF** not *PCTS_timer_create* **THEN**
TEST: A call to *timer_create*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for timer_create: PASS, NO_OPTION

14.2.3 Delete a Per-Process Timer

Function: *timer_delete*()

14.2.3.1 Synopsis

1

M_GA_stdC_proto_decl(int; timer_delete; timer_t timerid; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for timer_delete: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(timer_delete; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for timer_delete: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; timer_delete; time.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for timer_delete: PASS, NO_OPTION

4

M_GA_macro_args (timer_delete; time.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for timer_delete: PASS, NO_OPTION

14.2.3.2 Description

timer_delete
IF *PCTS_timer_delete* **THEN**
IF *PCTS_timer_create* **THEN**
TEST: A successful call to *timer_delete(timerid)* deletes the specified timer, *timerid*, previously created by the *timer_create*() function, and returns the value zero.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_TEST_SUPPORT, NO_OPTION

5 **IF** *PCTS_timer_delete* **THEN**
TEST: When the timer is armed when *timer_delete*() is called, the behavior is as if the timer is automatically disarmed before removal.
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_OPTION

D_1 IF *PCTS_timer_delete* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the disposition of pending signals for the deleted timer, does so in §14.2.3.2.
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_OPTION

D_2 IF *PCTS_timer_delete* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *timer_delete*() function does so in §14.2.3.2.
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_OPTION

14.2.3.3 Returns

R_1 IF *PCTS_timer_delete* **THEN**
TEST: When a call to *timer_delete*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion timer_delete in §14.2.3.4

R_2 IF *PCTS_timer_delete* **THEN**
TEST: When a call to *timer_delete*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §14.2.3.4

14.2.3.4 Errors

6 **IF** *PCTS_timer_delete* **THEN**
TEST: A call to *timer_delete(timerid)*, when the timer ID specified by *timerid* is not a valid timer ID, returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid timer ID or indicates that there is no way to generate an invalid timer ID on the system.
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_OPTION

7 **IF** not *PCTS_timer_delete* **THEN**
TEST: A call to *timer_delete*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for timer_delete: PASS, NO_OPTION

14.2.4 Per-Process Timers

Functions: *timer_settime*(), *timer_gettime*(), *timer_getoverrun*()

14.2.4.1 Synopsis

1
M_GA_stdC_proto_decl(int; timer_settime; timer_t timerid, int flags, const struct itimerspec *value, struct itimerspec *ovalue; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for timer_settime: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(timer_settime; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for timer_settime: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; timer_settime; time.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for timer_settime: PASS, NO_OPTION

4
M_GA_macro_args (timer_settime; time.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for timer_settime: PASS, NO_OPTION

5
M_GA_stdC_proto_decl(int; timer_gettime; timer_t timerid, struct itimerspec *value; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for timer_gettime: PASS[5, 6], NO_OPTION

6
M_GA_commonC_int_result_decl(timer_gettime; timer_t timerid, struct itimerspec *value; time.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3

Conformance for timer_gettime: PASS[5, 6], NO_OPTION

7

M_GA_macro_result_decl(int; timer_gettime; time.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for timer_gettime: PASS, NO_OPTION

8

M_GA_macro_args (timer_gettime; time.h;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for timer_gettime: PASS, NO_OPTION

9

M_GA_stdC_proto_decl(int; timer_getoverrun; , timer_t timerid; time.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for timer_getoverrun: PASS[9, 10], NO_OPTION

10

M_GA_commonC_int_result_decl(timer_getoverrun;,timer_t timerid; time.h;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for timer_getoverrun: PASS[9, 10], NO_OPTION

11

M_GA_macro_result_decl(int; timer_getoverrun; time.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for timer_getoverrun: PASS, NO_OPTION

12

M_GA_macro_args (timer_getoverrun; time.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for timer_getoverrun: PASS, NO_OPTION

14.2.4.2 Description

timer_settime

IF *PCTS_timer_settime* **THEN**
TEST: A successful call to *timer_settime(timerid, flags, value, ovalue)*, sets the time until the next expiration of the timer specified by *timerid* from the *it_value* member of the *value* is nonzero, and returns the value zero.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

13 **IF** *PCTS_timer_settime* **THEN**
TEST: When the specified timer is already armed when *timer_settime(timerid, flags, value, ovalue)* is called, this call resets the time until next expiration to the *value* specified.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

14 **IF** *PCTS_timer_settime* **THEN**
TEST: In a successful call to *timer_settime(timerid, flags, value, ovalue)*, if the *it_value* member of *value* is zero, the timer is disarmed.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

D_1 **IF** *PCTS_timer_settime* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the effect of disarming or resetting a timer on pending expiration notifications, does so in §14.2.4.2.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

15 **IF** *PCTS_timer_settime* **THEN**
SETUP: Include the header `<timer.h>`.
TEST: The constant TIMER_ABSTIME is defined.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

16 **IF** *PCTS_timer_settime* **THEN**
TEST: When the flag TIMER_ABSTIME is not set in the argument *flags*, *timer_settime(timerid, flags, value, ovalue)* behaves as if the time until next expiration is set to be equal to the interval specified by the *it_value* member of *value*; that is, the timer expires in *it_value* nanoseconds (see POSIX.1b {3} §14.1.1) from when the call is made.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

17 **IF** *PCTS_timer_settime* **THEN**
TEST: When the flag TIMER_ABSTIME is not set in the argument *flags*, *timer_settime(timerid, flags, value, ovalue)* behaves as if the time until next expiration is set to be equal to the difference between the absolute time specified by the *it_value* member of *value* and the current value of the clock associated with *timerid*; that is, the timer expires when the clock reaches the value specified by the *it_value* member of *value.*
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

18 **IF** *PCTS_timer_settime* **THEN**
TEST: When the flag TIMER_ABSTIME is set in the argument *flags,* and the specified time has already passed, *timer_settime(timerid, flags, value, ovalue)* succeeds and the expiration notification is made.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

19 **IF** *PCTS_timer_settime* **THEN**
TEST: A successful call to *timer_settime(timerid, flags, value, ovalue*), with a zero value of *it_interval* specifies a nonperiodic (one-time) timer.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

20 **IF** *PCTS_timer_settime* **THEN**
TEST: A successful call to *timer_settime(timerid, flags, value, ovalue*), with a nonzero *it_interval* specifies a periodic (or repetitive) timer, with the reload value of the timer set to the value specified by the *it_interval* member of *value.*
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

21 **IF** *PCTS_timer_settime* **THEN**
TEST: In calls to *timer_settime*(), time values that are between two consecutive nonnegative integer multiples of the resolution of the specified timer are founded up to the larger multiple of the resolution; quantization error does not cause the timer to expire earlier than the rounded time value.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

22 **IF** *PCTS_timer_settime* **THEN**
TEST: When the timer *timerid* is armed and the argument *ovalue* is not **NULL**, the call *timer_settime(timerid, flags, value, ovalue)* stores, in the location referenced by *ovalue,* a value representing the previous amount of time before the timer would have expired together with the previous timer reload value.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

23 **IF** *PCTS_timer_settime* **THEN**
TEST: When the timer *timerid* is disarmed and the argument *ovalue* is not **NULL**, the call *timer_settime(timerid, flags, value, ovalue)* stores, in the location referenced by *ovalue*, a value representing zero, together with the previous timer reload value.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

24 **IF** *PCTS_timer_settime* **THEN**
IF *PCTS_timer_gettime* **THEN**
TEST: In the call *timer_settime(timerid, flags, value, ovalue)*, the members of *ovalue* are subject to the resolution of the timer, and they are the same values that would be returned by a *timer_gettime*() call at the point in the time.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

timer_gettime
IF *PCTS_timer_gettime* **THEN**
TEST: A successful call to *timer_gettime*() stores both the reload value of the timer and the amount of time until the specified timer, *timerid*, expires into the space pointed to by the *value* argument; and returns the value zero.

The *it_value* member of this structure contains the amount of time before the timer expires, or zero if the timer is disarmed.

The *it_interval* member of *value* contains the reload value last set by *timer_settime*().
ELSE *NO_OPTION*
Conformance for timer_gettime: PASS, NO_OPTION

25 **IF** *PCTS_timer_gettime* **THEN**
TEST: The amount of time before the timer expires is returned as the interval until timer expiration, even if the timer is armed with absolute time.
ELSE *NO_OPTION*
Conformance for timer_gettime: PASS, NO_OPTION

26 **IF** {_POSIX_REALTIME_SIGNALS}**THEN**
TEST: Only a single signal is queued to the process for a given timer at any point in time.
ELSE *NO_TEST_SUPPORT*
Conformance for timer_settime: PASS, NO_TEST_SUPPORT

27 **IF** {_POSIX_REALTIME_SIGNALS}**THEN**
TEST: When a timer for which a signal is still pending expires, no signal is queued, a timer overrun occurs.
ELSE *NO_TEST_SUPPORT*

Conformance for timer_settime: PASS, NO_TEST_SUPPORT

timer_getoverrun
IF *PCTS_timer_getoverrun* **THEN**
IF {_POSIX_REALTIME_SIGNALS}**THEN**
TEST: When a timer expiration signal is delivered to a process, the *timer_getoverrun*() function returns the timer expiration overrun count for the specified timer.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

28 **IF** *PCTS_timer_getoverrun* **THEN**

TEST: The PCD.1b documents the value of {DELAYTIMER_MAX}, in §14.2.4.2.
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_OPTION

29 **IF** *PCTS_timer_getoverrun* **THEN**
TEST: The overrun count returned by *timer_getoverrun*() contains the number of extra timer expirations that occurred between the time the signal was generated (queued) and when it was delivered.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_TEST, NO_OPTION

30 **IF** *PCTS_timer_getoverrun* **THEN**
IF {DELAYTIMER_MAX} <= *PCTS_DELAYTIMER_MAX* **THEN**
TEST: When the number of extra expirations is greater than or equal to {DELAYTIMER_MAX}, then *timer_getoverrun*() returns {DELAYTIMER_MAX}.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

31 **IF** *PCTS_timer_getoverrun* **THEN**
TEST: The value returned by *timer_getoverrun*() applies to the most recent expiration signal delivery for the timer.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_TEST, NO_OPTION

D_2 **IF** *PCTS_timer_getoverrun* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the meaning of the overrun count returned, if no expiration signal has been delivered for the timer, or if {_POSIX_REALTIME_SIGNALS} is not supported, does so in §14.2.4.2.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_OPTION

D_3 **IF** *PCTS_timer_settime* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *timer_settime*() function does so in §14.2.4.2.
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_OPTION

D_4 **IF** *PCTS_timer_gettime* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *timer_gettime*() function does so in §14.2.4.2.
ELSE *NO_OPTION*
Conformance for timer_gettime: PASS, NO_OPTION

D_5 **IF** *PCTS_timer_getoverrun* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *timer_getoverrun*() function does so in §14.2.4.2.
ELSE *NO_OPTION*
Conformance for timer_getoverrun: PASS, NO_OPTION

14.2.4.3 Returns

R_1 **IF** *PCTS_timer_settime* **THEN**

TEST: When a call to *timer_settime*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion timer_settime in §14.2.4.4

R_2 IF *PCTS_timer_gettime* **THEN**
TEST: When a call to *timer_gettime*() completes successfully, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion timer_gettime in §14.2.4.4

R_3 IF *PCTS_timer_settime* **THEN**
TEST: When a call to *timer_settime*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
timer_settime()
ELSE *NO_OPTION*
SEE: All assertions in §14.2.4.4 controlled by *timer_settime*()

R_4 IF *PCTS_timer_gettime* **THEN**
TEST: When a call to *timer_gettime*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
timer_gettime()
ELSE *NO_OPTION*
SEE: All assertions in §14.2.4.4 controlled by *timer_gettime*()

R_5 IF *PCTS_timer_getoverrun* **THEN**
TEST: When a call to *timer_getoverrun*() completes successfully, the interface returns the timer expiration overrun count as explained in POSIX.1b {3} §14.2.4.2.
ELSE *NO_OPTION*
SEE: Assertion timer_getoverrun in §14.2.4.4

14.2.4.4 Errors

32 IF *PCTS_timer_settime* **THEN**
IF *PCTS_timer_create* and *PCTS_timer_delete* **THEN**
TEST: A call to *timer_settime*(), when the *timerid* argument does not correspond to an ID returned by *timer_create*() but not yet deleted by *timer_delete*(), returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid timer ID or indicates that there is no way to generate an invalid timer ID on the system.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

33 IF *PCTS_timer_gettime* **THEN**
IF *PCTS_timer_create* and *PCTS_timer_delete* **THEN**
TEST: A call to *timer_gettime*(), when the *timerid* argument does not correspond to an ID returned by *timer_create*() but not yet deleted by *timer_delete*(), returns a value of -1 and sets *errno* to [EINVAL].
NOTE: A subroutine is recommended that either returns an invalid timer ID or indicates that there is no way to generate an invalid timer ID on the system.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for timer_gettime: PASS, NO_TEST_SUPPORT, NO_OPTION

34 IF *PCTS_timer_getoverrun* **THEN**
IF *PCTS_timer_create* and *PCTS_timer_delete* **THEN**

TEST: A call to *timer_getoverrun(, timerid)* when the *timerid* argument does not correspond to an ID returned by *timer_create*() but not yet deleted by *timer_delete*(), returns a value of -1 and sets *errno* to [EINVAL].

NOTE: A subroutine is recommended that either returns an invalid timer ID or indicates that there is no way to generate an invalid timer ID on the system.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for timer_getoverrun: PASS, NO_TEST_SUPPORT, NO_OPTION

35 **IF** not *PCTS_timer_settime* **THEN**

TEST: A call to *timer_settime*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_OPTION

36 **IF** not *PCTS_timer_gettime* **THEN**

TEST: A call to *timer_gettime*() returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for timer_gettime: PASS, NO_OPTION

37 **IF** not *PCTS_timer_getoverrun* **THEN**

TEST: A call to *timer_getoverrun* returns a value of -1 and sets *errno* to [ENOSYS].

ELSE *NO_OPTION*

Conformance for timer_getoverrun: PASS, NO_OPTION

38 **IF** *PCTS_timer_settime* **THEN**

TEST: A call to *timer_settime(timerid, flags, value, ovalue*), when the *value* structure specifies a nanosecond value less than zero, returns a value of -1 and sets *errno* to [EINVAL].

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_OPTION

39 **IF** *PCTS_timer_settime* **THEN**

IF {INT_MAX}<= 10e9 **THEN**

TEST: A call to *timer_settime(timerid, flags, value, ovalue*), when the *value* structure specifies a nanosecond greater than or equal to 1000 million, returns a value of -1 and sets *errno* to [EINVAL].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for timer_settime: PASS, NO_TEST_SUPPORT, NO_OPTION

14.2.5 High Resolution Sleep

Function: *nanosleep*()

14.2.5.1 Synopsis

1

M_GA_stdC_proto_decl(int; nanosleep; const struct timespec *rmtp; time.h;;;)

SEE: Assertion GA_stdC_proto_decl in §2.7.3

Conformance for nanosleep: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(nanosleep; time.h;;;)

SEE: Assertion GA_commonC_int_result_decl in §2.7.3

Conformance for nanosleep: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; nanosleep; time.h;;;)

SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for nanosleep: PASS, NO_OPTION

4
M_GA_macro_args (nanosleep; time.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for nanosleep: PASS, NO_OPTION

14.2.5.2 Description

nanosleep
IF *PCTS_nanosleep* **THEN**
TEST: A successful call to *nanosleep(rqtp, rmtp)* causes the current process to be suspended from execution until the time interval specified by the *rqtp* argument has elapsed, whereupon it returns zero.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

5 **IF** *PCTS_nanosleep* **THEN**
TEST: In a successful call to *nanosleep(rqtp, rmtp)*, the suspension time is not less than the time specified by the *rqtp*, as measured by the system clock, CLOCK_REALTIME.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

6 **IF** *PCTS_nanosleep* **THEN**
TEST: In a successful call to *nanosleep(rqtp, rmtp)*, the suspension time may be longer than requested because the argument value is rounded up to an integer multiple of the sleep resolution.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_TEST, NO_OPTION

7 **IF** *PCTS_nanosleep* **THEN**
TEST: In a successful call to *nanosleep(rqtp, rmtp)*, the suspension time may be longer than requested because of the scheduling of other activity by the system.
NOTE: There is no known portable test method for this assertion.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_TEST, NO_OPTION

8 **IF** *PCTS_nanosleep* **THEN**
TEST:
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

9 **IF** *PCTS_nanosleep* **THEN**
TEST: The use of the *nanosleep*() function has no effect on the action or blockage of any signal.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

D_1 **IF** *PCTS_nanosleep* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *nanosleep*() function does so in §14.2.5.2.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

14.2.5.3 Returns

R_1 IF *PCTS_nanosleep* **THEN**
TEST: When a call to *nanosleep*() returns because the requested time has elapsed, the interface returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion nanosleep in §14.2.5.4

R_2 IF *PCTS_nanosleep* **THEN**
TEST: When a call to *nanosleep*() returns because it has been interrupted by a signal, the interface returns a value of -1, and sets *errno* to indicate the interruption.
ELSE *NO_OPTION*
SEE: All assertions in §14.2.5.4

10 IF *PCTS_nanosleep* **THEN**
TEST: When the *rmtp* argument is non-**NULL**, the *timespec* structure referenced by it is updated by *nanosleep(rqtp, rmtp)* to contain the amount of time remaining in the interval (the requested time minus the time actually slept).
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

11 IF *PCTS_nanosleep* **THEN**
TEST: When the *rmtp* argument is **NULL**, the remaining time is not returned by *nanosleep(rqtp, rmtp).*
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

R_3 IF *PCTS_nanosleep* **THEN**
TEST: When a call to *nanosleep*() completes unsuccessfully, the interface returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §14.2.5.4

14.2.5.4 Errors

12 IF *PCTS_nanosleep* **THEN**
TEST: A call to *nanosleep*(), when the *nanosleep*() function is interrupted by a signal, returns a value of -1 and sets *errno* to [EINTR].
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

13 IF *PCTS_nanosleep* **THEN**
TEST: A call to *nanosleep(rqtp, rmtp)*, when the *rqtp* argument specifies a nanosecond value less than zero, returns a value -1 and sets *errno* to [EINVAL].
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

14 IF *PCTS_nanosleep* **THEN**
IF INT_MAX<=10e9 **THEN**
TEST: A call to *nanosleep(rqtp, rmtp)*, when the *rqtp* argument specifies a nanosecond value greater than or equal to 1000 million, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_TEST_SUPPORT, NO_OPTION

15 IF not *PCTS_nanosleep* **THEN**
TEST: A call to *nanosleep*() returns a value of -1 and sets *errno* to [ENOSYS}.
ELSE *NO_OPTION*
Conformance for nanosleep: PASS, NO_OPTION

This page is intentionally blank.

Section 15: Message Passing

15.1 Data Definitions for Message Queues

D_1 SETUP: Include the header <MQUEUE.H>.
TEST: A PCD.1b that documents the symbols allowed by this standard to be in the headers <SYS/TYPES.H> <FCNT1.H> <TIME.H> <SIGNAL.H> are visible does so in §15.1.
ELSE *NO_OPTION*
Conformance for mq_intro: PASS, NO_OPTION

15.1.1 Data Structures

1 **SETUP:** Include the header <mqueue.h>.
TEST: The types *mqd_t* and *struct sigevent* are defined.
Conformance for mq_hdr: PASS

D_1 TEST: The PCD.1b documents the definition of *mqd_t* and *struct sigevent*, in §15.1.1.
Conformance for mq_hdr: PASS

3 **SETUP:** Include the header <mqueue.h>.
TEST: The structure *mq_attr* is defined and has at least the following members.

Member Type	Member Name	Description
long	*mq_flags*	Message queue flags.
long	*mq_maxmsg*	Maximum number of messages.
long	*mq_msgsize*	Maximum message size.
long	*mq_curmsgs*	Number of messages currently queued.

Conformance for mq_hdr: PASS

D_2 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents extensions to *mq_attr*, as permitted in POSIX.1b{3} §1.3.1.1 item (2), does so in §15.1.1.
ELSE *NO_OPTION*
Conformance for mq_hdr: PASS, NO_OPTION

4 **SETUP:** Include the header <mqueue.h>.
TEST: Extensions to *mq_attr* that may change the behavior of the application with respect to this standard when those fields in the structure are uninitialized, are enabled as required by POSIX.1b {3} §1.3.1.1.
NOTE: The corresponding statement in IEEE Std 1003.1b-1993 is not specific enough to write a portable test.
Conformance for mq_hdr: PASS, NO_TEST

D_3 IF a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the existence of flags other than O_NONBLOCK, does so in §15.1.1.
ELSE *NO_OPTION*
Conformance for mq_hdr: PASS, NO_OPTION

M_GA_mqOpenMaxFD () =
IF *PCTS_MQ_FILE_DESCRIPTORS* **THEN**
IF ({OPEN_MAX} <= *PCTS_OPEN_MAX*) and *PCTS_mq_open* **THEN**
TEST: A process calling *mq_open*() can simultaneously open a combination of files and message queues totaling at least {OPEN_MAX}.
TR: Test for opening {OPEN_MAX} message queues.

Test for opening a message queue after opening {OPEN_MAX}-1 files.

Test for opening a file after opening {OPEN_MAX}-1 message queues.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*

GA_mqOpenMaxFD
FOR: *mq_open*()
M_GA_mqOpenMaxFD()
Conformance for mq_hdr: PASS, NO_OPTION

M_GA_mqPCTSOpenMaxFD () =
IF *PCTS_MQ_FILE_DESCRIPTORS* **THEN**
IF ({OPEN_MAX} > *PCTS_OPEN_MAX*) and *PCTS_mq_open* **THEN**
TEST: A process calling *mq_open*() can simultaneously open a combination of files and message queues totaling at least *PCTS_OPEN_MAX*.
TR: Test for opening *PCTS_OPEN_MAX* message queues.

Test for opening a message queue after opening *PCTS_OPEN_MAX*-1 files.

Test for opening a file after opening *PCTS_OPEN_MAX-1* message queues.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*

GA_mqPCTSOpenMaxFD
FOR: *mq_open*()
M_GA_mqPCTSOpenMaxFD()
Conformance for mq_hdr: PASS, NO_OPTION

15.2 Message Passing Functions

15.2.1 Open a Message Queue

Function: *mq_open*()

15.2.1.1 Synopsis

1
M_GA_stdC_proto_decl(mqd_t; mq_open; const char *name, int oflag, ... oflag; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for mq_open: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(mqd_t; mq_open; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3

Conformance for mq_open: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(mqd_t; mq_open; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for mq_open: PASS, NO_OPTION

4

M_GA_macro_args (mq_open; mqueue.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for mq_open: PASS, NO_OPTION

15.2.1.2 Description

M_GA_mqOpenMaxFD()
SEE: Assertion GA_mqOpenMaxFD in §15.1.1.
Conformance for mq_open: PASS[OpenMaxMqs, PCTSOpenMaxMqs]

M_GA_mqPCTSOpenMaxFD()
SEE: Assertion GA_mqPCTSOpenMaxFD in §15.1.1.
Conformance for mq_open: PASS[OpenMaxMqs, PCTSOpenMaxMqs]

mq_open
IF *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A successful call to *mq_open*() establishes a connection between a process and a message queue, *name*, and returns a message queue, descriptor which can be used by other functions to refer to that message queue.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 IF *PCTS_mq_open* and a PCD.1b documents the following **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A PCD.1b that documents whether the name appears in the file system and is visible to other functions that take pathnames as arguments does so in §15.2.1.2.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

5

M_GA_portableFilenames(mq_open)
SEE: Assertion GA_portableFilenames in §2.2.2.40
Conformance for mq_open: PASS, NO_OPTION

6

M_GA_upperLowerNames(mq_open)
SEE: Assertion GA_upperLowerNames in §2.2.2.40
Conformance for mq_open: PASS, NO_OPTION

7

M_GA_PRNoTrunc(mq_open)
SEE: Assertion GA_PRNoTrunc in §2.3.6
Conformance for mq_open: PASS, NO_OPTION

8

M_GA_PRNoTruncError(mq_open)
SEE: Assertion GA_PRNoTruncError in §2.2.2.40

Conformance for mq_open: PASS, NO_OPTION

9 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When *name* begins with the slash character, then processes calling *mq_open(name, oflag, ...)* with the same value of *name* refer to the same message queue object, as long as that name has not been removed.
TR: Try the interface both in the same process that created the queue and in another process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_mq_open* **THEN**
TEST: The PCD.1b documents the effect of calling *mq_open(name, oflag, ...)*, if *name* does not begin with the slash character in §15.2.1.2.
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

D_3 **IF** *PCTS_mq_open* **THEN**
TEST: The PCD.1b documents the interpretation of slash characters other than the leading slash character in *name* in §15.2.1.2.
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

R_1 **IF** *PCTS_mq_open* **THEN**
TEST: When the *name* argument is not the name of an existing message queue and O_CREAT is not set, *mq_open(name, oflag, ...)* fails and returns an error.
ELSE *NO_OPTION*
SEE: Assertion mq_open_ENOENT in §15.2.1.4

10 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: The access permission to receive messages or send messages as specified by *oflag*, is granted if the calling process would be granted read or write access, respectively, to an equivalently protected file.

Read and write access to files is determined as described in 2.3.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

11 **FOR:** mq_open
M_GA_AP_classAccess(mq_open)
SEE: Assertion GA_AP_class Access in §2.3.2
Conformance for mq_open: PASS

12 **FOR:** mq_open
M_GA_AP_overrideFileAccess(mq_open)
SEE: Assertion GA_AP_overrideFile Access in §2.3.2
Conformance for mq_open: PASS

13 **IF** *PCTS_mq_open* **THEN**
SETUP: Include the header `<mqueue.h>`.
TEST: The constants O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, and O_NONBLOCK are defined and have different values.
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST

14 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_mq_send* and *PCTS_mq_receive* and *PCTS_GAP_mq_open* **THEN**
TEST: When a message queue is opened with the call *mq_open(name, oflag, ...)* and the flag O_RDONLY is set in *oflag*, the process can use the returned message queue descriptor with *mq_receive*(), but not with *mq_send*().
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

15 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A message queue may be opened multiple times in the same or different processes with the call *mq_open(name, oflag, ...)* and the flag O_RDONLY set in *oflag.*
TR: Try the interface both in the same process that created the queue and in another process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

16 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_mq_open* and *PCTS_mq_send* and *PCTS_mq_receive* **THEN**
TEST: When a message queue is opened with the call *mq_open(name, oflag, ...)* and the flag O_WRONLY is set in *oflag*, the process can use the returned message queue descriptor with *mq_send*() but not *mq_receive*().
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

17 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A message queue may be opened multiple times in the same or different processes with the call *mq_open(name, oflag, ...)* and the flag O_WRONLY set in *oflag.*
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

18 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_mq_open* and *PCTS_mq_send* and *PCTS_mq_receive* **THEN**
TEST: When a message queue is opened with the call *mq_open(name, oflag, ...)* and the flag O_RDWR is set in *oflag*, the process can use any of the functions allowed for O_RDONLY and O_WRONLY.
TR: Try both *mq_send*() and *mq_receive*().
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

19 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When a message queue is opened with the call *mq_open(name, oflag, ...)* and the flag O_RDWR is set in *oflag*, it may be open multiple times in the same or different processes for sending messages.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

20 **IF** *PCTS_mq_open* **THEN**

IF *PCTS_GAP_mq_open* **THEN**
TEST: When a message queue is opened with the call *mq_open(name, oflag, mode, attr)* the named message queue does not already exist, and the flag O_CREAT is set in *oflag*, a message queue is created without any messages in it.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

21 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When *mq_open(name, oflag, mode, attr)* is called, with O_CREAT set and O_EXCL cleared, and the named message queue already exists, the O_CREAT flag has no effect.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

22 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When a new message queue is created with *mq_open*(), the user ID of the message queue is set to the effective user ID of the process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

23 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When a new message queue is created with *mq_open*(), the group ID of the message queue is set to the effective group ID of the process.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

24 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When a new message queue is created with *mq_open(name, oflag, ...)* , the "file permission bits" are set to the value of *mode.*
TR: Test for octal values 0000 through 0777.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

D_4 **IF** *PCTS_mq_open* **THEN**
TEST: The PCD.1b documents the effect when bits in *mode* other than file permission bits are set in §15.2.1.2.
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

D_5 **IF** *PCTS_mq_open* **THEN**
TEST: The PCD.1b documents the implementation-defined default message queue attributes with which message queues are created if *mq_open(name, oflag, mode, attr)* is called and *attr* is **NULL**, in §15.2.1.2.
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

25 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: When *attr* is non_**NULL** and the calling process has the appropriate privilege on *name*, the message queue *mq_maxmsg* and *mq_msgsize* attributes are set to the

values of the corresponding members in the *mq_attr* structure referred to by *attr*.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

R_2 IF *PCTS_mq_open* **THEN**

TEST: When *attr* is non_**NULL**, but the calling process does not have the appropriate privilege on *name*, the *mq_open*() function fails and returns an error without creating the message queue.

ELSE *NO_OPTION*

SEE: Assertion mq_open_EACCES2 in §15.2.1.4

R_3 IF *PCTS_mq_open* **THEN**

TEST: When O_EXCL and O_CREAT are set, *mq_open*() fails if the message queue *name* exists.

ELSE *NO_OPTION*

SEE: Assertion mq_open_EEXIST in §15.2.1.4

26 IF *PCTS_mq_open* **THEN**

IF *PCTS_GAP_mq_open* **THEN**

TEST: The check for the existence of the message queue and the creation of the message queue if it does not exist is atomic with respect to other processes executing *mq_open*() naming the same *name* with O_EXCL and O_CREAT set.

There is no known reliable test method for this assertion.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

D_6 IF *PCTS_mq_open* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents the result of calling *mq_open(name, oflag, ...)* if O_EXCL is set and O_CREAT is not set does so in §15.2.2.2.

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_OPTION

27 IF *PCTS_mq_open* **THEN**

IF *PCTS_GAP_mq_open* **THEN**

TEST: The setting of O_NONBLOCK is associated with the open message queue descriptor and determines whether a *mq_send*() or *mq_receive*() waits for resources or messages that are not currently available, or fails with *errno* set to [EAGAIN].

See *mq_send*() and *mq_receive*() for details.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

28 IF *PCTS_mq_open* **THEN**

IF *PCTS_GAP_mq_open* **THEN**

TEST: The *mq_open*() function does not add or remove messages from the queue.

TR: Open a message queue, send a single message, re-open the same queue, then try two successive reads.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

D_7 IF *PCTS_mq_open* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *mq_open*() function does so in §15.2.1.2.

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_OPTION

15.2.1.3 Returns

R_4 IF *PCTS_mq_open* **THEN**

TEST: When a call to *mq_open*() completes successfully, the function returns a message queue descriptor.

ELSE *NO_OPTION*

SEE: Assertion mq_open in §15.2.1.2

R_5 IF *PCTS_mq_open* **THEN**

TEST: When a call to *mq_open*() completes unsuccessfully, the function returns *(mqd_t)* -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: All assertions in §15.2.1.4

15.2.1.4 Errors

mq_open_EACCES1

IF *PCTS_mq_open* **THEN**

IF *PCTS_GAP_mq_open* **THEN**

TEST: A call to *mq_open(name, oflag, ...)*, when the message queue exists and the permissions specified by *oflag* are denied, returns a value of -1 and sets *errno* to [EACCES].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

mq_open_EACCES2

IF *PCTS_mq_open* **THEN**

IF *PCTS_RAP_mq_open* **THEN**

TEST: A call to *mq_open(name, oflag, ...)*, when the message queue does not exist and permission to create the message queue is denied, returns a value of -1 and sets *errno* to [EACCES].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

mq_open_EEXIT

IF *PCTS_mq_open* **THEN**

IF *PCTS_GAP_mq_open* **THEN**

TEST: A call to *mq_open(name, oflag, ...)*, when O_CREAT and O_EXCL are set and the named message queue already exists, returns a value of -1 and sets *errno* to [EEXIST].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

29 **IF** *PCTS_mq_open* **THEN**

IF *PCTS_GAP_mq_open* **THEN**

TEST: A call to *mq_open*() when the *mq_open*() operation is interrupted by a signal, returns a value of -1 and sets *errno* to [EINTR].

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

30 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open*(), when the *mq_open(name, oflag, ...)* operation is not supported for the given name , returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

D-8 **IF** *PCTS_mq_open* **THEN**
TEST: The PCD.1b documents under what circumstances this error may be returned in §15.2.1.4
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

31 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open(name, oflag, ...)*, when O_CREAT is specified in *oflag*, the value of *attr* is not **NULL**, and *mq_maxmsg* is less than or equal to zero, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

32 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open(name, oflag, ...)*, when O_CREAT is specified in *oflag*, the value of *attr* is not **NULL**, and *mq_msgsize* is less than or equal to zero, returns a value of -1 and sets *errno* to [EINVAL].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

33 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* and ({OPEN_MAX} $\leq$ *PCTS_OPEN_MAX)* **THEN**
TEST: A call to *mq_open* (), when too many message queue descriptors or file descriptors are currently in use by this process, returns a value of -1 and sets *errno* to [EMFILE].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

34 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* and {MQ_OPEN_MAX} <= *PCTS_MQ_ OPEN_MAX)* **THEN**
TEST: A call to *mq_open* (), when too many message queue descriptors or file descriptors are currently in use by this process, returns a value of -1 and sets *errno* to [EMFILE].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

35 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* and {_POSIX_NO_TRUNC} and ({PATH_MAX} <= *PCTS_PATH_MAX)* **THEN**
TEST: A call to *mq_open (name, oflag, ...)*, when the length of the *name* string exceeds {PATH_MAX}, while {_POSIX_NO_TRUNC} is in effect, returns a value of -1 and sets *errno* to [ENAMETOOLONG].

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

36 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* and {_POSIX_NO_TRUNC} and ({NAME_MAX} <= *PCTS_NAME_MAX)* **THEN**
TEST: A call to *mq_open (name, oflag, ...),* when a pathname component is longer than {NAME_MAX}, while {_POSIX_NO_TRUNC} is in effect, returns a value of -1 and sets *errno* to [ENAMETOOLONG].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

37 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open* (), when too many message queues are currently open in the system, returns a value of -1 and sets *errno* to [ENFILE].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

mq_open_ENOENT
IF *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open* (), when O_CREAT is not set and the named message queue does not exist, returns a value of -1 and sets *errno* to [ENOENT].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST_SUPPORT, NO_OPTION

38 **IF** *PCTS_mq_open* **THEN**
IF *PCTS_GAP_mq_open* **THEN**
TEST: A call to *mq_open* (), when there is insufficient space for the creation of the new message queue, returns a value of -1 and sets *errno* to [ENOSPC].
There is no known reliable test method for this assertion.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_TEST, NO_TEST_SUPPORT, NO_OPTION

39 **IF** not *PCTS_mq_open* **THEN**
TEST: A call to *mq_open* () returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_open: PASS, NO_OPTION

15.2.2 Close a Message Queue

Function: *mq_close*()

15.2.2.1 Synopsis

1
M_GA_stdC_proto_decl(int; mq_close; mqd_t mqdes; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for mq_close: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(mq_close; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for mq_close: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; mq_close; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for mq_close: PASS, NO_OPTION

4

M_GA_macro_args (mq_close; mqueue.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for mq_close: PASS, NO_OPTION

15.2.2.2 Description

mq_close

IF *PCTS_mq_close* **THEN**
TEST: A call to *mq_close*() removes the association between the message queue descriptor, *mqdes*, and its message queue, and returns a value of zero.
ELSE *NO_OPTION*
Conformance for mq_close: PASS, NO_OPTION

D_1 IF *PCTS_mq_close* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents the results of using this message queue descriptor after successful return from this *mq_close*() and until the return of this message queue descriptor from a subsequent *mq_open*() does so in §15.2.2.2.
ELSE *NO_OPTION*
Conformance for mq_close: PASS, NO_OPTION

5 **IF** *PCTS_mq_close* **THEN**
IF *PCTS_mq_notify* **THEN**
TEST: When the process has successfully attached a notification request to the message queue via *mqdes*, a call to *mq_close*() removes this attachment and makes the message queue available for another process to attach for notification.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_close: PASS,NO_TEST_SUPPORT, NO_OPTION

D_2 IF *PCTS_mq_close* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mq_close*() function does so in §15.2.2.2.
ELSE *NO_OPTION*
Conformance for mq_close: PASS, NO_OPTION

15.2.2.3 Returns

R_1 IF *PCTS_mq_close* **THEN**
TEST: When a call to *mq_close*() completes successfully, it returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion mq_close in §15.2.2.2

R_2 IF *PCTS_mq_close* **THEN**
TEST: When a call to *mq_close*() completes unsuccessfully, the function returns a value of -1 and sets *errno* to indicate the error.

ELSE *NO_OPTION*
SEE: All assertions in §15.2.2.4

15.2.2.4 Errors

6 **IF** *PCTS_mq_close* **THEN**
TEST: A call to *mq_close(mqdes),*when the *mqdes* argument is not a valid message queue descriptor, returns a value of -1 and sets *errno* to [EBADF].
ELSE *NO_OPTION*
Conformance for mq_close: PASS, NO_OPTION

7 **IF** not *PCTS_mq_close* **THEN**
TEST: A call to *mq_close*(), returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mq_close: PASS, NO_OPTION

15.2.3 Remove a Message Queue

Function: *mq_unlink*()

15.2.3.1 Synopsis

1
M_GA_stdC_proto_decl(int; mq_unlink; const char *name; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for mq_unlink: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(mq_unlink; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for mq_unlink: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; mq_unlink; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for mq_unlink: PASS, NO_OPTION

4
M_GA_macro_args (mq_unlink; mqueue.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for mq_unlink: PASS, NO_OPTION

15.2.3.2 Description

mq_unlink
IF *PCTS_mq_unlink* **THEN**
TEST: A successful call to *mq_link*() removes the message queue named by the pathname *name,* and returns a value of zero.
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_OPTION

R_1 IF *PCTS_mq_unlink* **THEN**
IF ***PCTS_****mq_open* and *PCTS_gap_mq_open* **THEN**
TEST: After a successful call to *mq_unlink(name)* a call to *mq_open(name, oflag, ...)* fails if the flag O_CREAT is not set in *flags.*
ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*
SEE: Assertion mq_unlink_ENOENT in §15.2.3.4

5 **IF** *PCTS_mq_unlink* **THEN**
TEST: When one or more processes have the message queue open when *mq_unlink*() is called, destruction of the message queue is postponed until all references to the message queue have been closed.
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_OPTION

D_1 **IF** *PCTS_mq_unlink* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mq_link*() function does so in §15.2.3.4.
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_OPTION

15.2.3.3 Returns

R_2 **IF** *PCTS_mq_unlink* **THEN**
TEST: When a call to *mq_link*() completes successfully, the function returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion mq_unlink in §15.2.3.2

R_3 **IF** *PCTS_mq_unlink* **THEN**
TEST: When a call to *mq_link*() completes unsuccessfully, the named message queue is unchanged and the function returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §15.2.3.4

15.2.3.4 Errors

6 **IF** *PCTS_mq_unlink* **THEN**
TEST: A call to *mq_unlink(name)*, when permission is denied to unlink the named message queue, returns a value of -1 and sets *errno* to [EACCES].
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_OPTION

7 **IF** *PCTS_mq_unlink* **THEN**
IF {_POSIX_NO_TRUNC} and ({NAME_MAX} <= *PCTS_NAME_MAX)* **THEN**
TEST: A call to *mq_unlink(name)*, when the length of the *name* string exceeds {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect, returns a value of -1 and sets *errno* to [ENAMETOOLONG].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_TEST_SUPPORT, NO_OPTION

mq_unlink_ENOENT
IF *PCTS_mq_unlink* **THEN**
TEST: A call to *mq_unlink(name)*, when the named message queue does not exist, returns a value of -1 and sets *errno* to [ENOENT].
ELSE *NO_OPTION*
Conformance for mq_unlink: PASS, NO_OPTION

8 **IF** not *PCTS_mq_unlink* **THEN**
TEST: A call to *mq_unlink*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*

Conformance for mq_unlink: PASS, NO_OPTION

15.2.4 Send a Message to a Message Queue

Function: *mq_send*()

15.2.4.1 Synopsis

1

M_GA_stdC_proto_decl(int; mq_send; mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for mq_send: PASS[1, 2], NO_OPTION

2

M_GA_commonC_int_result_decl(mq_send; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for mq_send: PASS[1, 2], NO_OPTION

3

M_GA_macro_result_decl(int; mq_send; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for mq_send: PASS, NO_OPTION

4

M_GA_macro_args (mq_send; mqueue.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for mq_send: PASS, NO_OPTION

15.2.4.2 Description

mq_send

IF *PCTS_mq_send* **THEN**
TEST: A successful call to *mq_send*() adds the message pointed to by the argument *msg_ptr* to the message queue specified by *mqdes*, and returns a value of zero.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

R_1 **IF** *PCTS_mq_send* **THEN**
TEST: When the value of *msg_len*, which specifies the length of the message in bytes pointed to by *msg_ptr*, is greater than the *mq_msgsize* attribute of the message queue, *mq_send*() fails.
ELSE *NO_OPTION*
SEE: Assertion mq_send_EMSGSIZE in §15.2.4.4

5 **IF** *PCTS_mq_send* **THEN**
TEST: When the specified message queue is not full, *mq_send*() behaves as if the message is inserted into the message queue at the position indicated by the *msg_prio* argument.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

6 **IF** *PCTS_mq_send* **THEN**
TEST: A message with a larger numeric value of *msg_prio* is inserted before messages with lower values of *msg_prio*.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

7 **IF** *PCTS_mq_send* **THEN**
TEST: A message is inserted after other messages in the queue, if any, with equal *msg_prio*.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

R_2 **IF** *PCTS_mq_send* **THEN**
TEST: The value of *msg_prio* is less than or equal to {MQ_PRIO_MAX}.
ELSE *NO_OPTION*
SEE: Assertion mq_send_EINVAL in §15.2.4.4

8 **IF** *PCTS_mq_send* **THEN**
TEST: When the specified message queue is full and O_NONBLOCK is not set in the message queue description associated with *mqdes*, *mq_send*() blocks until space becomes available to enqueue the message, or until *mq_send*() is interrupted by a signal.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

9 **IF** *PCTS_mq_send* **THEN**
IF {_POSIX_PRIORITY_SCHEDULING} **THEN**
TEST: When more than one process is waiting to send when space becomes available in the message queue and the {_POSIX_PRIORITY_SCHEDULING} option is supported, then the process of the highest priority that has been waiting the longest unblocked to send its message.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 **IF** *PCTS_mq_send* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents which waiting process is unblocked otherwise does so in §15.2.4.2.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

R_3 **IF** *PCTS_mq_send* **THEN**
TEST: When the specified message queue is full and O_NONBLOCK is set in the message queue description associated with *mqdes*, the message is not queued and *mq_send*() returns an error.
ELSE *NO_OPTION*
SEE: Assertion mq_send_EAGAIN in §15.2.4.4

D_2 **IF** *PCTS_mq_send* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mq_send*() function does so in §15.2.4.2.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

15.2.4.3 Returns

R_4 **IF** *PCTS_mq_send* **THEN**
TEST: When a call to *mq_send*() completes successfully, it returns a value of 0.
ELSE *NO_OPTION*
SEE: Assertion mq_send in §15.2.4.2

R_5 **IF** *PCTS_mq_send* **THEN**
TEST: When a call to *mq_send*() completes unsuccessfully, no message is enqueued, the function returns -1, and *errno* is set to indicate the error.

ELSE *NO_OPTION*
SEE: All assertions in §15.2.4.4

15.2.4.4 Errors

mq_send_EAGAIN
IF *PCTS_mq_send* **THEN**
TEST: A call to *mq_send(mqdes, msg_ptr, msg_len, msg_prio),* when the O_NONBLOCK flag is set in the message queue description associated with *mqdes*, and the specified message queue is full, returns a value of -1 and sets *errno* to [EAGAIN].
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

10 **IF** *PCTS_mq_send* **THEN**
TEST: A call to *mq_send(mqdes, msg_ptr, msg_len, msg_prio),* when the *mqdes* argument is not a valid message queue descriptor open for writing, returns a value of -1 and sets *errno* to [EBADF].
TR: Open a message queue O_RDONLY, then send to that queue.

Send to an invalid message queue.
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

11 **IF** *PCTS_mq_send* **THEN**
TEST: A call to *mq_send*(), when a signal interrupts the call, returns a value of -1 and sets *errno* to [EINTR].
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

mq_send_EINVAL
IF *PCTS_mq_send* **THEN**
IF {MQ_PRIO_MAX} < {UINT_MAX} **THEN**
TEST: A call to *mq_send(mqdes, msg_ptr, msg_len, msg_prio),* when the value of *msg_prio* is outside the valid range, returns a value of -1 and sets *errno* to [EINVAL].
TR: Try a value of {MQ_PRIO_MAX}+1, if it is less than {UINT_MAX}
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_TEST_SUPPORT, NO_OPTION

mq_send_EMSGSIZE
IF *PCTS_mq_send* **THEN**
TEST: A call to *mq_send(mqdes, msg_ptr, msg_len, msg_prio),* when the specified message length, *msg_len*, exceeds the message size attribute of the message queue, returns a value of -1 and sets *errno* to [EMSGSIZE].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

12 **IF** not *PCTS_mq_send* **THEN**
TEST: A call to *mq_send*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mq_send: PASS, NO_OPTION

15.2.5 Receive a Message From a Message Queue

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 §2.7.3
930 *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 §2.7.3
934 *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 §1.3.4
938 *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 §2.7.3
942 *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_OPTION*
951 *Conformance for mq_receive: PASS, NO_OPTION*

952 **5** **IF** *PCTS_mq_receive* **THEN**
953 **TEST:** When the size of the buffer in bytes, specified by the *msg_len* argument, is less than
954 the *mq_msgsize* attribute of the message queue, the function *mq_receive(mqdes,*
955 *msg_ptr, msg_len, msg_prio)* fails and returns an error.
956 **ELSE** *NO_OPTION*
957 *Conformance for mq_receive: PASS, NO_OPTION*

958 **6** **IF** *PCTS_mq_receive* **THEN**
959 **TEST:** When the argument *msg_prio* is not **NULL**, the priority of the selected message is
960 stored in the location referenced by *msg_prio*.
961 **ELSE** *NO_OPTION*
962 *Conformance for mq_receive: PASS, NO_OPTION*

963 **7** **IF** *PCTS_mq_receive* **THEN**
964 **TEST:** When the specified message queue is empty and O_NONBLOCK is not set in the
965 message queue description associated with *mqdes*, *mq_receive*() blocks until a
966 message is enqueued on the message queue or until *mq_receive*() is interrupted by a
967 signal.
968 **ELSE** *NO_OPTION*
969 *Conformance for mq_receive: PASS, NO_OPTION*

970 **8** **IF** *PCTS_mq_receive* **THEN**
971 **IF** {_POSIX_PRIORITY_SCHEDULING} **THEN**

TEST: When more than one process is waiting to receive a message when a message arrives at an empty queue and the {_POSIX_PRIORITY_SCHEDULING} option is supported, then the process of highest priority that has been waiting the longest is selected to receive the message.

ELSE *NO_TEST_SUPPORT*

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_TEST_SUPPORT, NO_OPTION

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

TEST: A PCD.1b that documents which waiting process receives the message if more than one process is waiting to receive a message when a message arrives at an empty queue and the {_POSIX_PRIORITY_SCHEDULING} option is not supported, does so in §15.2.5.2.

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_OPTION

R_1 IF *PCTS_mq_receive* **THEN**

TEST: When the specified message queue is empty and O_NONBLOCK is set in the message queue description associated with *mqdes*, no message is removed from the queue, and *mq_receive*() returns an error.

ELSE *NO_OPTION*

SEE: Assertion for mq_receive_EAGAIN in §15.2.5.4

D_2 IF *PCTS_mq_receive* and a PCD.1b documents the following **THEN**

TEST: A PCD.1b that documents whether or not it supports the *mq_receive*() function does so in §15.2.5.2.

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_OPTION

15.2.5.3 Returns

R_2 IF *PCTS_mq_receive* **THEN**

TEST: When a call to *mq_receive*() completes successfully, it returns the length of the selected message in bytes and the message is removed from the queue.

ELSE *NO_OPTION*

SEE: Assertion for mq_receive in §15.2.5.2

R_3 IF *PCTS_mq_receive* **THEN**

TEST: When a call to *mq_receive*() completes unsuccessfully, no message is removed from the queue, the function returns a value of -1, and sets *errno* to indicate the error.

ELSE *NO_OPTION*

SEE: All assertions in §15.2.5.4

15.2.5.4 Errors

mq_receive_EAGAIN

IF *PCTS_mq_receive* **THEN**

TEST: A test to *mq_receive(mqdes, msg_ptr, msg_len, msg_prio)*, when O_NONBLOCK is set in the message description associated with *mqdes*, and the specified message queue is empty, returns a value of -1 and sets *errno* to [EAGAIN].

ELSE *NO_OPTION*

Conformance for mq_receive: PASS, NO_OPTION

9 IF *PCTS_mq_receive* **THEN**

TEST: A call to *mq_receive(mqdes, msg_ptr, msg_len, msg_prio)*, when the *mqdes* argument is not a valid message queue descriptor open for reading, returns a value of -1 and sets *errno* to [EBADF].

TR: Open a message queue O_WRONLY, then receive from that queue.

Receive from an invalid message queue.
ELSE *NO_OPTION*
Conformance for mq_receive: PASS, NO_OPTION

10 **IF** *PCTS_mq_receive* **THEN**
TEST: A call to *mq_receive(mqdes, msg_ptr, msg_len, msg_prio)*, when the specified message buffer size, *msg_len*, is less than the message size attribute of the message queue, returns a value of -1 and sets *errno* to [EMSGSIZE].
ELSE *NO_OPTION*
Conformance for mq_receive: PASS, NO_OPTION

11 **IF** *PCTS_mq_receive* **THEN**
TEST: A call to *mq_receive*(), when operation is interrupted by a signal, returns a value of -1 and sets *errno* to [EINTR].
ELSE *NO_OPTION*
Conformance for mq_receive: PASS, NO_OPTION

12 **IF** not *PCTS_mq_receive* **THEN**
TEST: A call to *mq_receive*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mq_receive: PASS, NO_OPTION

13 **IF** *PCTS_mq_receive* **THEN**
IF *PCTS_DETECT_MESSAGE_DATA_CORRUPTION* **THEN**
TEST: A call to *mq_receive*(), when the implementation has detected a data corruption problem with the message, returns a value of -1 and sets *errno* to [EBADMSG].
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_receive: PASS, NO_TEST_SUPPORT, NO_OPTION

15.2.6 Notify Process that a Message is Available on a Queue

Function: *mq_notify*()

15.2.6.1 Synopsis

1
M_GA_stdC_proto_decl(int; mq_notify; mqd_t mqdes, const struct sigevent *notification; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for mq_notify: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(mq_notify; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for mq_notify: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; mq_notify; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for mq_notify: PASS, NO_OPTION

4
M_GA_macro_args (mq_notify; mqueue.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for mq_notify: PASS, NO_OPTION

15.2.6.2 Description

mq_notify

IF *PCTS_mq_notify* **THEN**
TEST: A call to *mq_notify(mqdes, notification)*, if the argument *notification* is not **NULL**, registers the calling process to be notified of message arrival at an empty message queue associated with the specified message queue descriptor, *mqdes*, and returns a value of zero.

The notification specified by the *notification* argument is sent to the process when the message queue transitions from empty to nonempty.
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

R_1 IF *PCTS_mq_notify* **THEN**
TEST: At any time, only one process may be registered for notification by a message queue. When the calling process, or any other process, has already registered for notification of message arrival at the specified message queue, subsequent attempts to register for that message queue fails.
ELSE *NO_OPTION*
SEE: Assertion mq_notify_EBUSY in §15.2.5.4

5 IF *PCTS_mq_notify* **THEN**
TEST: When *notification* is **NULL** and the process currently registered for notification by the specified message queue, the existing registration is removed.
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

6 IF *PCTS_mq_notify* **THEN**
TEST: When the notification is sent to the registered process, its registration is removed and the message queue is made available for registration.
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

7 IF *PCTS_mq_notify* **THEN**
IF *PCTS_mq_receive* **THEN**
TEST: When a process has registered for notification of message arrival at a message queue, and some process is blocked in *mq_receive*() waiting to receive a message when a message arrives at the queue, the arriving message satisfies the appropriate *mq_receive*() (see POSIX.1b {3} §15.2.5).

The resulting behavior is as if the message queue remains empty, and no notification is sent.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_TEST_SUPPORT, NO_OPTION

D_1 IF *PCTS_mq_notify* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mq_notify*() function does so in §15.2.6.2.
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

15.2.6.3 Returns

R_2 IF *PCTS_mq_notify* **THEN**
TEST: When a call to *mq_notify*() completes successfully, it returns a value of 0.
ELSE *NO_OPTION*

SEE: Assertion mq_notify in §15.2.6.2

R_3 IF *PCTS_mq_notify* **THEN**
TEST: When a call to *mq_notify*() completes unsuccessfully, it returns a value of -1, and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §15.2.6.4

15.2.6.4 Errors

8 **IF** *PCTS_mq_notify* **THEN**
TEST: A call to *mq_notify(mqdes, notification*), when the *mqdes* argument is not a valid message queue descriptor, returns a value of -1 and sets *errno* to [EBADF].
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

mq_notify_EBUSY
IF *PCTS_mq_notify* **THEN**
TEST: A call to *mq_notify*(), when a process is already registered for notification by the message queue, returns a value of -1 and sets *errno* to [EBUSY].
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

9 **IF** not *PCTS_mq_notify* **THEN**
TEST: A call to *mq_notify*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mq_notify: PASS, NO_OPTION

15.2.7 Set Message Queue Attributes

Function: *mq_setattr*()

15.2.7.1 Synopsis

1
M_GA_stdC_proto_decl(int; mq_setattr; mqd_t mqdes, const struct mq_attr *mqstat, struct mq_attr *omqstat; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for mq_setattr: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(mq_setattr; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for mq_setattr: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; mq_setattr; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for mq_setattr: PASS, NO_OPTION

4
M_GA_macro_args (mq_setattr; mqueue.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for mq_setattr: PASS, NO_OPTION

15.2.7.2 Description

mq_setattr

IF *PCTS_mq_setattr* **THEN**
TEST: A call to *mq_setattr*() sets attributes associated with the message queue specified by *mqdes* and returns a value of zero.
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

5 **IF** *PCTS_mq_setattr* **THEN**
TEST: The message queue attributes corresponding to *mq_flags*, defined in the *mq_attr* structure are set to the specified values upon successful completion of *mq_setattr*().
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

D_1 **IF** *PCTS_mq_setattr* **THEN**
TEST: The PCD.1b documents any implementation-defined flags that can be set in *mq_flags* in §15.2.7.2.
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

6 **IF** *PCTS_mq_setattr* **THEN**
TEST: The values of the *mq_maxmsg*, *mq_msgsize*, and *mg_curmsgs* members of the *mq_attr* structure are ignored by *mq_setattr*().
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

7 **IF** *PCTS_mq_setattr* **THEN**
IF *PCTS_mq_getattr* **THEN**
TEST: When *omqstat* is non-**NULL**, the function *mq_setattr*() stores, in the location referenced by *omqstat*, the previous message queue attributes and the current queue status.

These values are the same as would be returned by a call to *mq_getattr*() at that point.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_TEST_SUPPORT, NO_OPTION

D_2 **IF** *PCTS_mq_setattr* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mq_setattr*() function does so in §15.2.7.2.
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

15.2.7.3 Returns

R_1 **IF** *PCTS_mq_setattr* **THEN**
TEST: When a call to *mq_setattr*() completes successfully, the function returns a value of 0 and changes the attributes of the message queue as specified.
ELSE *NO_OPTION*
SEE: Assertion mq_setattr in §15.2.7.2

R_2 **IF** *PCTS_mq_setattr* **THEN**
TEST: When a call to *mq_setattr*() completes unsuccessfully, the message queue attributes are unchanged, and the function returns a value of -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*

SEE: All assertions in §15.2.7.4

15.2.7.4 Errors

9 **IF** not *PCTS_mq_setattr* **THEN**
TEST: A call to *mq_setattr*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mq_setattr: PASS, NO_OPTION

15.2.8 Get Message Queue Attributes

Function: *mq_getattr*()

15.2.8.1 Synopsis

1
M_GA_stdC_proto_decl(int; mq_getattr; mqd_t mqdes, struct mq_attr *mqstat; mqueue.h;;;)
SEE: Assertion GA_stdC_proto_decl in §2.7.3
Conformance for mq_getattr: PASS[1, 2], NO_OPTION

2
M_GA_commonC_int_result_decl(mq_getattr; mqueue.h;;;)
SEE: Assertion GA_commonC_int_result_decl in §2.7.3
Conformance for mq_getattr: PASS[1, 2], NO_OPTION

3
M_GA_macro_result_decl(int; mq_getattr; mqueue.h;;;)
SEE: Assertion GA_macro_result_decl in §1.3.4
Conformance for mq_getattr: PASS, NO_OPTION

4
M_GA_macro_args (mq_getattr; mqueue.h;;;)
SEE: Assertion GA_macro_args in §2.7.3
Conformance for mq_getattr: PASS, NO_OPTION

15.2.8.2 Description

mq_getattr
IF *PCTS_mq_getattr* **THEN**
TEST: A call to *mq_getattr*() gets status information and attributes associated with the message queue specified in *mqdes* and returns zero.
ELSE *NO_OPTION*
Conformance for mq_getattr: PASS, NO_OPTION

5 **IF** *PCTS_mq_getattr* **THEN**
IF *PCTS_mq_setattr* **THEN**
TEST: After a successful call to *mq_getattr(mqdes, mqstat)* the *mq_flags* member within the *mq_attr* structure referenced by the *mqstat* argument has the value that was set when the message queue was created but with modifications made by subsequent *mq_setattr*() calls.
ELSE *NO_TEST_SUPPORT*
ELSE *NO_OPTION*
Conformance for mq_getattr: PASS, NO_TEST_SUPPORT, NO_OPTION

6 **IF** *PCTS_mq_getattr* **THEN**

TEST: After a successful call to *mq_getattr(mqdes, mqstat)* the *mq_maxmsg* and the *mq_msgsize* members within the *mq_attr* structure referenced by the *mqstat* argument have the values that were set at message queue creation.
ELSE *NO_OPTION*
Conformance for mq_getattr: PASS, NO_OPTION

7 **IF** *PCTS_mq_getattr* **THEN**
TEST: After a successful call to *mq_getattr(mqdes, mqstat)* the *mq_curmsgs* member within the *mq_attr* structure referenced by the *mqstat* argument has the value that is set according to the number of messages currently on the queue.
ELSE *NO_OPTION*
Conformance for mq_getattr: PASS, NO_OPTION

D_1 **IF** *PCTS_mq_getattr* and a PCD.1b documents the following **THEN**
TEST: A PCD.1b that documents whether or not it supports the *mq_getattr*() function does so in §15.2.8.2.
ELSE *NO_OPTION*
Conformance for mq_getattr: PASS, NO_OPTION

15.2.8.3 Returns

R_1 **IF** *PCTS_mq_getattr* **THEN**
TEST: When a call to *mq_getattr*() completes successfully, it returns 0.
ELSE *NO_OPTION*
SEE: Assertion mq_getattr in §15.2.8.2

R_2 **IF** *PCTS_mq_getattr* **THEN**
TEST: When a call to *mq_getattr*() completes unsuccessfully, the function returns -1 and sets *errno* to indicate the error.
ELSE *NO_OPTION*
SEE: All assertions in §15.2.8.4

8 **IF** *PCTS_mq_getattr* **THEN**
TEST:
ELSE *NO_OPTION*
Conformance for mq_getattr: PASS, NO_OPTION

15.2.8.4 Errors

9 **IF** *PCTS_mq_getattr* **THEN**
TEST: A call to *mq_getattr(mqdes, mqstat)*, when the *mqdes* argument is not a valid message queue descriptor, returns a value of -1 and sets *errno* to [EBADF].
ELSE *NO_OPTION*
Conformance for mq_getattr: PASS, NO_OPTION

10 **IF** not *PCTS_mq_getattr* **THEN**
TEST: A call to *mq_getattr*() returns a value of -1 and sets *errno* to [ENOSYS].
ELSE *NO_OPTION*
Conformance for mq_getattr: PASS, NO_OPTION

Annex A
(normative)

Conforming Test Results

A.1 General

Assertions for conformance

Subclause	Assertion ID	Conforming Results
1.3.3.3	GD_CommonC_diffs	General Documentation Assertion, no test results
1.3.4	GA_macro_args	General Assertion, no test results
1.3.4	GA_macro_result_decl	General Assertion, no test results

A.2 Terminology and General Requirements

Assertions for definitions

Subclause	Assertion ID	Conforming Results
2.2.2.40	GA_portableFilenames	General Assertion, no test results
2.2.2.40	GA_upperLowerNames	General Assertion, no test results
2.2.2.105	D_1	PASS
2.2.2.105	D_2	PASS
2.2.2.108	R_1	Reference Assertion, no test results
2.2.2.109	R_2	Reference Assertion, no test results
2.2.2.119	GA_syncIODataIntegrityRead	General Assertion, no test results
2.2.2.119	GA_syncIODataIntegrityWbeforeR	General Assertion, no test results
2.2.2.119	GA_syncIODataIntegrityWrite	General Assertion, no test results
2.2.2.120	GA_syncIOFileIntegrityRead	General Assertion, no test results
2.2.2.120	GA_syncIOFileIntegrityWrite	General Assertion, no test results
2.2.2.126	D_3	PASS

Assertions for General Concepts

Subclause	Assertion ID	Conforming Results
2.3.2	GA_AP_overrideFileAccess	General Assertion, no test results
2.3.2	GA_AP_overrideExecAccess	General Assertion, no test results
2.3.2	GA_AP_classAccess	General Assertion, no test results

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**

227 Subclause	Assertion ID	Conforming Results
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**

232 Subclause	Assertion ID	Conforming Results
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_EnvironCaseSensitive	General Assertion, no test results
241 2.6	GA_EnvironPortNames	General Assertion, no test results

242 **Assertions for C Language Definitions**

243 Subclause	Assertion ID	Conforming Results
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**

260	Subclause	Assertion ID	Conforming Results
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**

267	Subclause	Assertion ID	Conforming Results
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**

280	Subclause	Assertion ID	Conforming Results
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	**Subclause**	**Assertion ID**	**Conforming Results**
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	**Subclause**	**Assertion ID**	**Conforming Results**
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 execle**

329	**Subclause**	**Assertion ID**	**Conforming Results**
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

Assertions for execlp

Subclause	Assertion ID	Conforming Results
3.1.2.2	1	PASS, NO_OPTION
3.1.2.2	2	PASS, NO_OPTION
3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	D_1	PASS, NO_OPTION
3.1.2.2	4	PASS, NO_OPTION, NO_TEST
3.1.2.2	5	PASS, NO_OPTION
3.1.2.2	D_2	PASS, NO_OPTION
3.1.2.2	6	PASS, NO_OPTION, NO_TEST
3.1.2.2	7	PASS, NO_OPTION
3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	10	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_3	PASS, NO_OPTION
3.1.2.2	11	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_4	PASS, NO_OPTION

Assertions for execvp

Subclause	Assertion ID	Conforming Results
3.1.2.2	1	PASS, NO_OPTION
3.1.2.2	2	PASS, NO_OPTION
3.1.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	D_1	PASS, NO_OPTION
3.1.2.2	4	PASS, NO_OPTION, NO_TEST
3.1.2.2	5	PASS, NO_OPTION
3.1.2.2	D_2	PASS, NO_OPTION
3.1.2.2	6	PASS, NO_OPTION, NO_TEST
3.1.2.2	7	PASS, NO_OPTION
3.1.2.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
3.1.2.2	10	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_3	PASS, NO_OPTION
3.1.2.2	11	PASS, NO_OPTION, NO_TEST
3.1.2.2	D_4	PASS, NO_OPTION

Assertions for Process Termination

Subclause	Assertion ID	Conforming Results
No assertions for Process Termination		

Assertions for wait

Subclause	Assertion ID	Conforming Results
No assertions for wait		

Assertions for _exit

Subclause	Assertion ID	Conforming Results
3.2.2.2	1	PASS, NO_OPTION, NO_TEST_SUPPORT
3.2.2.2	2	PASS, NO_OPTION, NO_TEST_SUPPORT
3.2.2.2	3	PASS, NO_OPTION, NO_TEST_SUPPORT
3.2.2.2	4	PASS, NO_OPTION, NO_TEST
3.2.2.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
3.2.2.2	6	PASS, NO_OPTION, NO_TEST
3.2.2.2	D_1	PASS, NO_OPTION
3.2.2.2	D_2	PASS

Assertions for signal.h

Subclause	Assertion ID	Conforming Results
3.3.1.1	1	PASS
3.3.1.1	2	PASS, NO_OPTION
3.3.1.1	3	PASS
3.3.1.1	4	PASS
3.3.1.1	5	PASS
3.3.1.1	D_1	PASS
3.3.1.2	6	PASS
3.3.1.2	D_2	PASS, NO_OPTION
3.3.1.2	7	PASS
3.3.1.2	D_3	PASS, NO_OPTION
3.3.1.2	GA_sigev_value	General Assertion, no test results
3.3.1.2	GA_sigqueueValue	General Assertion, no test results
3.3.1.2	GA_sigPending Queued	General Assertion, no test results
3.3.1.2	D_4	PASS, NO_OPTION
3.3.1.2	ga_queuedAndRegularSignals	General Assertion, no test results
3.3.1.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.3	12	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.3	13	PASS, NO_OPTION
3.3.1.3	14	PASS, NO_OPTION, NO_TEST
3.3.1.3	D_5	PASS, NO_OPTION
3.3.1.3	15	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.1.3	16	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
3.3.1.3	17	PASS, NO_OPTION, NO_TEST_SUPPORT`

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

Assertions for kill

Subclause	Assertion ID	Conforming Results
No assertions for kill		

Assertions for sigemptyset

Subclause	Assertion ID	Conforming Results
No assertions for sigemptyset		

Assertions for sigfillset

Subclause	Assertion ID	Conforming Results
No assertions for sigfillset		

Assertions for sigaddset

Subclause	Assertion ID	Conforming Results
No assertions for sigaddset		

Assertions for sigdelset

Subclause	Assertion ID	Conforming Results
No assertions for sigdelset		

Assertions for sigismember

Subclause	Assertion ID	Conforming Results
No assertions for sigismember		

Assertions for sigaction

Subclause	Assertion ID	Conforming Results
3.3.4.2	1	PASS
3.3.4.2	2	PASS, NO_OPTION
3.3.4.2	3	PASS
3.3.4.2	4	PASS
3.3.4.2	D_1	PASS
3.3.4.2	5	PASS, NO_OPTION
3.3.4.4	6	PASS, NO_OPTION

Assertions for sigprocmask

Subclause	Assertion ID	Conforming Results
3.3.5.2	D_1	PASS, NO_OPTION

Assertions for sigpending

Subclause	Assertion ID	Conforming Results
No assertions for sigpending		

Assertions for sigsuspend

Subclause	Assertion ID	Conforming Results
No assertions for sigsuspend		

Assertions for sigwaitinfo

Subclause	Assertion ID	Conforming Results
3.3.8.1	1	PASS[1,2], NO_OPTION
3.3.8.1	2	PASS[1,2], NO_OPTION
3.3.8.1	3	PASS, NO_OPTION
3.3.8.1	4	PASS, NO_OPTION
3.3.8.2	9	PASS, NO_OPTION
3.3.8.2	10	PASS, NO_OPTION
3.3.8.2	D_1	PASS, NO_OPTION
3.3.8.2	11	PASS, NO_OPTION
3.3.8.2	12	PASS, NO_OPTION
3.3.8.2	13	PASS, NO_OPTION
3.3.8.2	14	PASS, NO_OPTION, NO_TEST
3.3.8.2	D_2	PASS, NO_OPTION
3.3.8.2	15	PASS, NO_OPTION
3.3.8.2	18	PASS, NO_OPTION

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

3.3.9.2	D_2	PASS, NO_OPTION
3.3.9.3	R_1	Reference Assertion, no test results
3.3.9.3	R_2	Reference Assertion, no test results
3.3.9.4	13	PASS, NO_OPTION, NO_TEST
3.3.9.4	14	PASS, NO_OPTION
3.3.9.4	15	PASS, NO_OPTION, NO_TEST
3.3.9.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.9.4	17	PASS, NO_OPTION
3.3.9.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT
3.3.9.4	19	PASS, NO_OPTION

Assertions for intro_timer_ops

Subclause	Assertion ID	Conforming Results
No assertions for intro_timer_ops		

Assertions for alarm

Subclause	Assertion ID	Conforming Results
No assertions for alarm		

Assertions for pause

Subclause	Assertion ID	Conforming Results
No assertions for pause		

Assertions for sleep

Subclause	Assertion ID	Conforming Results
No assertions for sleep		

A.4 Process Environment

Assertions for sysconf

Subclause	Assertion ID	Conforming Results
4.8.1.2	1	PASS
4.8.1.2	2	PASS, NO_TEST_SUPPORT
4.8.1.2	3	PASS, NO_TEST_SUPPORT
4.8.1.2	4	PASS, NO_TEST_SUPPORT
4.8.1.2	5	PASS, NO_TEST_SUPPORT

A.5 Files and Directories

Assertions for intro5

Subclause	Assertion ID	Conforming Results
No assertions for sleep		

Assertions for dirent.h

Subclause	Assertion ID	Conforming Results
No assertions for dirent.h		

Assertions for directory operations

Subclause	Assertion ID	Conforming Results
No assertions for directory operations		

Assertions for chdir

Subclause	Assertion ID	Conforming Results
No assertions for chdir		

Assertions for getcwd

Subclause	Assertion ID	Conforming Results
No assertions for getcwd		

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

Assertions for creat

Subclause	Assertion ID	Conforming Results
No assertions for creat		

Assertions for umask

Subclause	Assertion ID	Conforming Results
No assertions for umask		

Assertions for link

Subclause	Assertion ID	Conforming Results
No assertions for link		

Assertions for mkdir

Subclause	Assertion ID	Conforming Results
No assertions for mkdir		

Assertions for mkfifo

Subclause	Assertion ID	Conforming Results
No assertions for mkfifo		

Assertions for unlink

Subclause	Assertion ID	Conforming Results
No assertions for unlink		

Assertions for rmdir

Subclause	Assertion ID	Conforming Results
No assertions for rmdir		

Assertions for rename

Subclause	Assertion ID	Conforming Results
No assertions for rename		

Assertions for stat.h

Subclause	Assertion ID	Conforming Results
5.6.1.1	D_1	PASS, NO_OPTION
5.6.1.1	1	PASS
5.6.1.1	2	PASS, NO_TEST_SUPPORT
5.6.1.1	3	PASS, NO_TEST_SUPPORT
5.6.1.1	4	PASS, NO_TEST_SUPPORT
5.6.1.1	5	PASS, NO_TEST_SUPPORT
5.6.1.1	6	PASS, NO_TEST_SUPPORT
5.6.1.1	7	PASS, NO_TEST_SUPPORT
5.6.1.1	8	PASS, NO_TEST_SUPPORT
5.6.1.1	9	PASS, NO_TEST_SUPPORT
5.6.1.1	10	PASS, NO_TEST_SUPPORT

Assertions for fstat

Subclause	Assertion ID	Conforming Results
5.6.2.2	1	PASS

Assertions for stat

Subclause	Assertion ID	Conforming Results
No assertions for stat		

Assertions for access

Subclause	Assertion ID	Conforming Results
No assertions for access		

Assertions for fchmod

Subclause	Assertion ID	Conforming Results
5.6.4.1	1	PASS[1, 2], NO_OPTION
5.6.4.1	2	PASS[1, 2], NO_OPTION
5.6.4.1	3	PASS, NO_OPTION
5.6.4.1	4	PASS, NO_OPTION
5.6.4.2	5	PASS, NO_OPTION
5.6.4.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
5.6.4.2	7	PASS, NO_OPTION

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_SUPPORT1
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

5.7.1.2	2	PASS, NO_OPTION
5.7.1.2	3	PASS
5.7.1.2	4	PASS, NO_OPTION
5.7.1.2	5	PASS
5.7.1.2	6	PASS, NO_OPTION
5.7.1.2	7	PASS
5.7.1.2	D_1	PASS, NO_OPTION

Assertions for fpathconf

Subclause	Assertion ID	Conforming Results
5.7.1.2	2	PASS, NO_OPTION
5.7.1.2	3	PASS
5.7.1.2	4	PASS, NO_OPTION
5.7.1.2	5	PASS
5.7.1.2	6	PASS, NO_OPTION
5.7.1.2	7	PASS
5.7.1.2	D_1	PASS, NO_OPTION

A.6 Input and Output Primitives

Assertions for intro6

Subclause	Assertion ID	Conforming Results
No assertions for intro6		

Assertions for pipe

Subclause	Assertion ID	Conforming Results
No assertions for pipe		

Assertions for dup

Subclause	Assertion ID	Conforming Results
No assertions for dup		

Assertions for dup2

Subclause	Assertion ID	Conforming Results
No assertions for dup2		

Assertions for close

Subclause	Assertion ID	Conforming Results
6.3.1.2	1	PASS, NO_OPTION, NO_TEST
6.3.1.2	2	PASS, NO_OPTION, NO_TEST
6.3.1.2	D_1	PASS
6.3.1.2	3	PASS, NO_OPTION
6.3.1.2	4	PASS, NO_OPTION

Assertions for read

Subclause	Assertion ID	Conforming Results
6.4.1.2	1	PASS, NO_OPTION, NO_TEST
6.4.1.2	2	PASS, NO_OPTION, NO_TEST
6.4.1.2	3	PASS, NO_OPTION, NO_TEST
6.4.1.2	D_1	PASS, NO_OPTIONS

Assertions for write

Subclause	Assertion ID	Conforming Results
6.4.2.2	1	PASS, NO_OPTION, NO_TEST
6.4.2.2	2	PASS, NO_OPTION, NO_TEST
6.4.2.2	D_1	PASS, NO_OPTION

Assertions for fcntl.h

Subclause	Assertion ID	Conforming Results
6.5.1	1	PASS

Assertions for fcntl

Subclause	Assertion ID	Conforming Results
6.5.2.2	4	PASS
6.5.2.2	5	PASS
6.5.2.2	6	PASS
6.5.2.2	7	PASS
6.5.2.2	8	PASS
6.5.2.2	9	PASS
6.5.2.2	10	PASS
6.5.2.2	D_1	PASS, NO_OPTION
6.5.2.4	41	PASS
6.5.2.4	42	PASS, NO_TEST_SUPPORT
6.5.2.4	42.1	PASS, NO_TEST_SUPPORT
6.5.2.4	42.2	PASS, NO_OPTION
6.5.2.4	48	PASS, NO_TEST_SUPPORT
6.5.2.4	49	PASS, NO_TEST_SUPPORT
6.5.2.4	49.1	PASS, NO_TEST_SUPPORT

Assertions for lseek

Subclause	Assertion ID	Conforming Results
6.5.3.2	D_1	PASS, NO_OPTION

Assertions for syncio

Subclause	Assertion ID	Conforming Results
6.6	D_1	PASS

Assertions for fsync

Subclause	Assertion ID	Conforming Results
6.6.1.1	1	PASS[1, 2], NO_OPTION
6.6.1.1	2	PASS[1, 2], NO_OPTION
6.6.1.1	3	PASS, NO_OPTION
6.6.1.1	4	PASS, NO_OPTION
6.6.1.2	D_1	PASS, NO_OPTION
6.6.1.2	5	PASS, NO_OPTION, NO_TEST
6.6.1.2	R_1	Reference Assertion, no test results
6.6.1.2	D_2	PASS, NO_OPTION
6.6.1.2	6	PASS, NO_OPTION, NO_TEST
6.6.1.2	7	PASS, NO_OPTION, NO_TEST
6.6.1.3	R_2	Reference Assertion, no test results
6.6.1.3	8	PASS
6.6.1.4	9	PASS, NO_OPTION
6.6.1.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
6.6.1.4	11	PASS, NO_OPTION
6.6.1.4	12	PASS, NO_OPTION, NO_TEST

Assertions for fdatasync

Subclause	Assertion ID	Conforming Results
6.6.2.1	1	PASS[1,2], NO_OPTION
6.6.2.1	2	PASS[1,2], NO_OPTION
6.6.2.1	3	PASS, NO_OPTION
6.6.2.1	4	PASS, NO_OPTION
6.6.2.2	5	PASS, NO_OPTION, NO_TEST
6.6.2.2	R_1	Reference Assertion, no test results
6.6.2.2	6	PASS, NO_OPTION, NO_TEST
6.6.2.2	7	PASS, NO_OPTION, NO_TEST
6.6.2.3	8	PASS
6.6.2.4	9	PASS, NO_OPTION
6.6.2.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
6.6.2.4	11	PASS, NO_OPTION
6.6.2.4	12	PASS, NO_OPTION, NO_TEST

Assertions for aio.h

Subclause	Assertion ID	Conforming Results
6.7.1.1	1	PASS
6.7.1.1	D_1	PASS
6.7.1.1	2	PASS, NO_TEST
6.7.1.1	3	PASS, NO_OPTION
6.7.1.1	4	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.1.1	D_2	PASS
6.7.1.1	5	PASS, NO_OPTION
6.7.1.1	6	PASS, NO_OPTION
6.7.1.1	7	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.1.1	8	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.1.1	D_3	PASS, NO_OPTION

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	einval1	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

6.7.4.4	R_8	Reference Assertion, no test results
6.7.4.4	28	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	lio_write_ebadf1	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	lio_write_ebadf2	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	lio_write_einval1	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	29	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	30	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	31	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	32	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.4.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	35	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	R_10	Reference Assertion, no test results
6.7.4.4	R_11	Reference Assertion, no test results
6.7.4.4	36	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.4.4	37	PASS, NO_OPTION
6.7.4.4	38	PASS, NO_OPTION
6.7.4.4	39	PASS, NO_OPTION
6.7.4.4	40	PASS, NO_OPTION

Assertions for aio_error

Subclause	Assertion ID	Conforming Results
6.7.5.1	1	PASS[1, 2], NO_OPTION
6.7.5.1	2	PASS[1, 2], NO_OPTION
6.7.5.1	3	PASS, NO_OPTION
6.7.5.1	4	PASS, NO_OPTION
6.7.5.2	5	PASS, NO_OPTION
6.7.5.2	R_1	Reference Assertion, no test results
6.7.5.2	6	PASS, NO_OPTION
6.7.5.3	7	PASS, NO_OPTION
6.7.5.3	R_2	Reference Assertion, no test results
6.7.5.3	8	PASS, NO_OPTION
6.7.5.4	9	PASS, NO_OPTION
6.7.5.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.5.4	11	PASS, NO_OPTION

Assertions for aio_return

Subclause	Assertion ID	Conforming Results
6.7.6.1	1	PASS[1, 2], NO_OPTION
6.7.6.1	2	PASS[1, 2], NO_OPTION
6.7.6.1	3	PASS, NO_OPTION
6.7.6.1	4	PASS, NO_OPTION
6.7.6.2	return_status	PASS, NO_OPTION
6.7.6.2	R_1	Reference Assertion, no test results
6.7.6.2	D_1	PASS, NO_OPTION
6.7.6.2	R_2	Refernce Assertion, no test results
6.7.6.2	5	PASS, NO_OPTION

6.7.6.3	R_3	Reference Assertion, no test results
6.7.6.3	D_2	PASS, NO_OPTION
6.7.6.4	einval1	PASS, NO_OPTION
6.7.6.4	6	PASS, NO_OPTION
6.7.6.4	7	PASS, NO_OPTION

Assertions for aio_cancel

Subclause	Assertion ID	Conforming Results
6.7.7.1	1	PASS[1, 2], NO_OPTION
6.7.7.1	2	PASS[1, 2], NO_OPTION
6.7.7.1	3	PASS, NO_OPTION
6.7.7.1	4	PASS, NO_OPTION
6.7.7.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.2	R_1	Reference Assertion, no test results
6.7.7.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.7.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.2	D_1	PASS, NO_OPTION
6.7.7.2	D_2	PASS, NO_OPTION
6.7.7.3	R_2	Reference Assertion, no test results
6.7.7.3	10	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.7.3	11	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.3	R_3	Reference Assertion, no test results
6.7.7.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.7.4	13	PASS, NO_OPTION

Assertions for aio_suspend

Subclause	Assertion ID	Conforming Results
6.7.8.1	1	PASS[1, 2], NO_OPTION
6.7.8.1	2	PASS[1, 2], NO_OPTION
6.7.8.1	3	PASS, NO_OPTION
6.7.8.1	4	PASS, NO_OPTION
6.7.8.2	completion	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.8.2	interrupt	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.8.2	timeout	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.8.2	already_completed	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.8.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.8.2	D_1	PASS, NO_OPTION
6.7.8.2	R_1	Reference Assertion, no test results
6.7.8.3	R_2	Reference Assertion, no test results
6.7.8.3	R_3	Reference Assertion, no test results
6.7.8.4	R_4	Reference Assertion, no test results
6.7.8.4	6	PASS
6.7.8.4	no_support	PASS, NO_OPTION

Assertions of aio_fsync

Subclause	Assertion ID	Conforming Results
6.7.9.1	1	PASS[1, 2], NO_OPTION
6.7.9.1	2	PASS[1, 2], NO_OPTION
6.7.9.1	3	PASS, NO_OPTION
6.7.9.1	4	PASS, NO_OPTION
6.7.9.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.9.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.9.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.9.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.9.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
6.7.9.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.2	D_1	PASS, NO_OPTION
6.7.9.3	R_1	Reference Assertion, no test results
6.7.9.3	R_2	Reference Assertion, no test results
6.7.9.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.4	15	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT
6.7.9.4	18	PASS, NO_OPTION
6.7.9.4	19	PASS, NO_OPTION

A.7 Device- and Class-Specific Functions

Assertions for Section_7

Subclause	Assertion ID	Conforming Results
No assertions for Section_7		

A.8 Language-Specific Services for the C Programming Language

Subclause	Assertion ID	Conforming Results
8.2.2.2	D_1	PASS, NO_OPTION

A.9 System Databases

Assertions for Section_9

Subclause	Assertion ID	Conforming Results
No assertions for Section_9		

A.10 Data Interchange Format

Assertions for Section_10

Subclause	Assertion ID	Conforming Results
No assertions for Section_10		

A.11 Synchronization

Assertions for sem_hdr

Subclause	Assertion ID	Conforming Results
11.1	1	PASS
11.1	GA_semOpenMaxFD	General Assertion, no test results
11.1	GA_semPCTSOpenMaxFD	General Assertion, no test results

Assertions for sem_init

Subclause	Assertion ID	Conforming Results
11.2.1.1	1	PASS[1, 2], NO_OPTION
11.2.1.1	2	PASS[1, 2], NO_OPTION
11.2.1.1	3	PASS, NO_OPTION
11.2.1.1	4	PASS, NO_OPTION
11.2.1.2	OpenMaxSems	PASS[OpenMaxSems, PCTSOpenMaxSems]
11.2.1.2	PCTS OpenMaxSems	PASS[OpenMaxSems, PCTSOpenMaxSems]
11.2.1.2	sem_init	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.2	D_1	PASS, NO_OPTION
11.2.1.2	D_2	PASS, NO_OPTION
11.2.1.2	D_3	PASS, NO_OPTION
11.2.1.3	R_1	Reference Assertion, no test results
11.2.1.3	R_2	Reference Assertion, no test results
11.2.1.4	8	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	9	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
11.2.1.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	11	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.1.4	15	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for sem-destroy

Subclause	Assertion ID	Conforming Results
11.2.2.1	1	PASS[1, 2], NO_OPTION
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**

1176	Subclause	Assertion ID	Conforming Results
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**

1192	Subclause	Assertion ID	Conforming Results
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

11.2.5.4	9	PASS, NO_OPTION
11.2.5.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.5.4	11	PASS, NO_OPTION
11.2.5.4	12	PASS, NO_OPTION

Assertions for sem_wait

Subclause	Assertion ID	Conforming Results
11.2.6.1	1	PASS[1, 2], NO_OPTION
11.2.6.1	2	PASS[1, 2], NO_OPTION
11.2.6.1	3	PASS, NO_OPTION
11.2.6.1	4	PASS, NO_OPTION
11.2.6.2	sem_wait	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	12	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	14	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	D_1	PASS, NO_OPTION
11.2.6.3	15	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.3	R_1	Reference Assertion, no test results
11.2.6.3	R_3	Reference Assertion, no test results
11.2.6.4	19	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	21	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	23	PASS, NO_OPTION
11.2.6.4	25	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for sem_trywait

Subclause	Assertion ID	Conforming Results
11.2.6.1	5	PASS[5, 6], NO_OPTION
11.2.6.1	6	PASS[5, 6], NO_OPTION
11.2.6.1	7	PASS, NO_OPTION
11.2.6.1	8	PASS, NO_OPTION
11.2.6.2	sem_trywait	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	11	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	13	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.2	D_2	PASS, NO_OPTION
11.2.6.3	16	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.3	R_2	Reference Assertion, no test results
11.2.6.3	R_4	Reference Assertion, no test results
11.2.6.4	17	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	18	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	20	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	22	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.6.4	24	PASS, NO_OPTION
11.2.6.4	26	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for sem_post

Subclause	Assertion ID	Conforming Results
11.2.7.1	1	PASS[1, 2], NO_OPTION
11.2.7.1	2	PASS[1, 2], NO_OPTION
11.2.7.1	3	PASS, NO_OPTION
11.2.7.1	4	PASS, NO_OPTION
11.2.7.2	sem_post	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	R_1	Reference Assertion, no test results
11.2.7.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	D_1	PASS, NO_OPTION
11.2.7.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.2	D_2	PASS, NO_OPTION
11.2.7.3	R_2	Reference Assertion, no test results
11.2.7.3	R_3	Reference Assertion, no test results
11.2.7.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.7.4	11	PASS, NO_OPTION

Assertions for sem_getvalue

Subclause	Assertion ID	Conforming Results
11.2.8.1	1	PASS[1, 2], NO_OPTION
11.2.8.1	2	PASS[1, 2], NO_OPTION
11.2.8.1	3	PASS, NO_OPTION
11.2.8.1	4	PASS, NO_OPTION
11.2.8.2	sem_getvalue	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.8.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.8.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.8.2	D_1	PASS, NO_OPTION
11.2.8.3	R_1	Reference Assertion, no test results
11.2.8.3	R_2	Reference Assertion, no test results
11.2.8.4	7	PASS, NO_OPTION, NO_TEST_SUPPORT
11.2.8.4	8	PASS, NO_OPTION

A.12 Memory Management

Assertions for mem-intro

Subclause	Assertion ID	Conforming Results
12	1	PASS
12	D_1	PASS, NO_OPTION
12	D_2	PASS, NO_OPTION
12	D_3	PASS, NO_OPTION
12	R_1	Reference Assertion, no test results
12	R_2	Reference Assertion, no test results
12	R_3	Reference Assertion, no test results

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**

1301	Subclause	Assertion ID	Conforming Results
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**

1327	Subclause	Assertion ID	Conforming Results
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

12.2.1.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
12.2.1.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	35	PASS, NO_OPTION
12.2.1.4	36	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	37	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	mmap_ENOTSUP	PASS, NO_OPTION
12.2.1.4	38	PASS, NO_OPTION
12.2.1.4	39	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	40	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.1.4	41	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for munmap

Subclause	Assertion ID	Conforming Results
12.2.2.1	1	PASS[1, 2], NO_OPTION
12.2.2.1	2	PASS[1, 2], NO_OPTION
12.2.2.1	3	PASS, NO_OPTION
12.2.2.1	4	PASS, NO_OPTION
12.2.2.2	munmap	PASS, NO_OPTION
12.2.2.2	munmap_SIGSEGV	PASS, NO_OPTION
12.2.2.2	5	PASS, NO_OPTION
12.2.2.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.2.2	D_1	PASS, NO_OPTION
12.2.2.2	munlock_remove_maps	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.2.2	D_2	PASS, NO_OPTION
12.2.2.2	D_3	PASS, NO_OPTION
12.2.2.3	R_1	Reference Assertion, no test results
12.2.2.3	R_2	Reference Assertion, no test results
12.2.2.4	8	PASS, NO_OPTION, NO_TEST
12.2.2.4	9	PASS, NO_OPTION
12.2.2.4	10	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for mprotect

Subclause	Assertion ID	Conforming Results
12.2.3.1	1	PASS[1, 2], NO_OPTION
12.2.3.1	2	PASS[1, 2], NO_OPTION
12.2.3.1	3	PASS, NO_OPTION
12.2.3.1	4	PASS, NO_OPTION
12.2.3.2	mprotect	PASS, NO_OPTION
12.2.3.2	R_1	Reference Assertion, no test results
12.2.3.2	5	PASS, NO_OPTION
12.2.3.2	R_2	Reference Assertion, no test results
12.2.3.2	D_1	PASS, NO_OPTION
12.2.3.2	6	PASS, NO_OPTION
12.2.3.2	7	PASS, NO_OPTION
12.2.3.2	8	PASS, NO_OPTION
12.2.3.2	9	PASS, NO_OPTION

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

Assertions for shm_open

Subclause	Assertion ID	Conforming Results
12.3.1.1	1	PASS[1, 2], NO_OPTION
12.3.1.1	2	PASS[1, 2], NO_OPTION
12.3.1.1	3	PASS, NO_OPTION
12.3.1.1	4	PASS, NO_OPTION
12.3.1.2	shm_open	PASS, NO_OPTION
12.3.1.2	5	PASS, NO_OPTION
12.3.1.2	D_1	PASS, NO_OPTION
12.3.1.2	6	PASS, NO_OPTION
12.3.1.2	7	PASS, NO_OPTION
12.3.1.2	8	PASS, NO_OPTION
12.3.1.2	9	PASS, NO_OPTION
12.3.1.2	10	PASS, NO_OPTION
12.3.1.2	D_2	PASS, NO_OPTION
12.3.1.2	D_3	PASS, NO_OPTION
12.3.1.2	11	PASS, NO_OPTION
12.3.1.2	12	PASS, NO_OPTION
12.3.1.2	D_4	PASS, NO_OPTION
12.3.1.2	13	PASS, NO_OPTION
12.3.1.2	14	PASS, NO_OPTION
12.3.1.2	15	PASS, NO_OPTION
12.3.1.2	16	PASS, NO_OPTION
12.3.1.2	17	PASS, NO_OPTION
12.3.1.2	18	PASS, NO_OPTION
12.3.1.2	19	PASS, NO_OPTION
12.3.1.2	20	PASS, NO_OPTION
12.3.1.2	21	PASS, NO_OPTION
12.3.1.2	22	PASS, NO_OPTION
12.3.1.2	23	PASS, NO_OPTION
12.3.1.2	24	PASS, NO_OPTION
12.3.1.2	D_5	PASS, NO_OPTION
12.3.1.2	25	PASS, NO_OPTION
12.3.1.2	26	PASS, NO_OPTION
12.3.1.2	R_1	Reference Assertion, no test results
12.3.1.2	27	PASS, NO_OPTION, NO_TEST
12.3.1.2	D_6	PASS, NO_OPTION
12.3.1.2	28	PASS, NO_OPTION
12.3.1.2	29	PASS, NO_OPTION
12.3.1.2	D_7	PASS, NO_OPTION
12.3.1.2	30	PASS, NO_OPTION
12.3.1.2	D_8	PASS, NO_OPTION
12.3.1.2	D_9	PASS, NO_OPTION
12.3.1.3	R_2	Reference Assertion, no test results
12.3.1.3	R_3	Reference Assertion, no test results
12.3.1.4	31	PASS, NO_OPTION
12.3.1.4	32	PASS, NO_OPTION

12.3.1.4	33	PASS, NO_OPTION
12.3.1.4	34	PASS, NO_OPTION
12.3.1.4	35	PASS, NO_OPTION
12.3.1.4	D_10	PASS, NO_OPTION
12.3.1.4	36	PASS, NO_OPTION
12.3.1.4	37	PASS, NO_OPTION, NO_TEST_SUPPORT
12.3.1.4	38	PASS, NO_OPTION, NO_TEST_SUPPORT
12.3.1.4	39	PASS, NO_OPTION
12.3.1.4	40	PASS, NO_OPTION
12.3.1.4	41	PASS, NO_OPTION, NO_TEST
12.3.1.4	42	PASS, NO_OPTION

Assertions for shm_unlink

Subclause	Assertion ID	Conforming Results
12.3.2.1	1	PASS[1, 2], NO_OPTION
12.3.2.1	2	PASS[1, 2], NO_OPTION
12.3.2.1	3	PASS, NO_OPTION
12.3.2.1	4	PASS, NO_OPTION
12.3.2.2	shm_unlink	PASS, NO_OPTION
12.3.2.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
12.3.2.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
12.3.2.2	D_1	PASS, NO_OPTION
12.3.2.3	R_1	Reference Assertion, no test results
12.3.2.3	R_2	Reference Assertion, no test results
12.3.2.4	7	PASS, NO_OPTION
12.3.2.4	8	PASS, NO_OPTION, NO_TEST_RESULTS
12.3.2.4	9	PASS, NO_OPTION
12.3.2.4	10	PASS, NO_OPTION

A.13 Execution Scheduling

Assertions for sched_param

Subclause	Assertion ID	Conforming Results
13.1	1	PASS
13.1	D_1	PASS, NO_OPTION
13.1	2	PASS, NO_TEST
13.1	3	PASS

Assertions for sched_policy

Subclause	Assertion ID	Conforming Results
13.2	1	PASS
13.2	2	PASS, NO_TEST
13.2	D_1	PASS, NO_OPTION
13.2	3	PASS
13.2.1	sched_fifo1	PASS, NO_OPTION, NO_TEST_SUPPORT

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**

Subclause	Assertion ID	Conforming Results
13.3.2.1	1	PASS[1, 2], NO_OPTION
13.3.2.1	2	PASS[1, 2], NO_OPTION
13.3.2.1	3	PASS, NO_OPTION
13.3.2.1	4	PASS, NO_OPTION
13.3.2.2	sched_getparam	PASS, NO_OPTION
13.3.2.2	5	PASS, NO_OPTION
13.3.2.2	6	PASS, NO_OPTION
13.3.2.2	D_1	PASS, NO_OPTION
13.3.2.2	D_2	PASS, NO_OPTION
13.3.2.3	R_1	Reference Assertion, no test results
13.3.2.3	R_2	Reference Assertion, no test results
13.3.2.4	7	PASS, NO_OPTION
13.3.2.4	8	PASS, NO_OPTION
13.3.2.4	9	PASS, NO_OPTION

Assertions for sched_setscheduler

Subclause	Assertion ID	Conforming Results
13.3.3.1	1	PASS[1, 2], NO_OPTION
13.3.3.1	2	PASS[1, 2], NO_OPTION
13.3.3.1	3	PASS, NO_OPTION
13.3.3.1	4	PASS, NO_OPTION
13.3.3.2	sched_setscheduler	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	D_1	PASS, NO_OPTION
13.3.3.2	6	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	D_2	PASS, NO_OPTION
13.3.3.2	D_3	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.2	D_4	PASS, NO_OPTION
13.3.3.2	R_1	Reference Assertion, no test results
13.3.3.2	D_5	PASS, NO_OPTION
13.3.3.3	R_2	Reference Assertion, no test results
13.3.3.3	R_3	Reference Assertion, no test results
13.3.3.4	11	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.4	12	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.4	13	PASS, NO_OPTION
13.3.3.4	14	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.4	15	PASS, NO_OPTION, NO_TEST_SUPPORT
13.3.3.4	16	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for sched_getscheduler

Subclause	Assertion ID	Conforming Results
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**

1706	**Subclause**	**Assertion ID**	**Conforming Results**
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**

1717	**Subclause**	**Assertion ID**	**Conforming Results**
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**

1732	**Subclause**	**Assertion ID**	**Conforming Results**
1733	13.3.6.1	5	PASS[5, 6], NO_OPTION

13.3.6.1	6	PASS[5, 6], NO_OPTION
13.3.6.1	7	PASS, NO_OPTION
13.3.6.1	8	PASS, NO_OPTION
13.3.6.2	sched_get_priority_min	PASS, NO_OPTION
13.3.6.2	15	PASS, NO_OPTION
13.3.6.2	D_2	PASS, NO_OPTION
13.3.6.3	R_2	Reference Assertion, no test results
13.3.6.3	R_4	Reference Assertion, no test results
13.3.6.4	17	PASS, NO_OPTION
13.3.6.4	22	PASS, NO_OPTION

Assertions for sched_rr_get_interval

Subclause	Assertion ID	Conforming Results
13.3.6.1	9	PASS[9, 10], NO_OPTION
13.3.6.1	10	PASS[9, 10], NO_OPTION
13.3.6.1	11	PASS, NO_OPTION
13.3.6.1	12	PASS, NO_OPTION
13.3.6.2	sched_rr_get_interval	PASS, NO_OPTION, NO_TEST
13.3.6.2	13	PASS, NO_OPTION, NO_TEST
13.3.6.2	D_3	PASS, NO_OPTION
13.3.6.2	R_5	Reference Assertion, no test results
13.3.6.3	R_6	Reference Assertion, no test results
13.3.6.4	20	PASS, NO_OPTION
13.3.6.4	23	PASS, NO_OPTION

A.14 Clocks and Timers

Assertions for timer_hdr

Subclause	Assertion ID	Conforming Results
14.1.1	1	PASS
14.1.1	D_1	PASS, NO_OPTION
14.1.1	2	PASS, NO_TEST
14.1.1	3	PASS
14.1.1	4	PASS, NO_TEST_SUPPORT
14.1.1	5	PASS
14.1.1	6	PASS
14.1.1	D_2	PASS, NO_OPTION
14.1.1	7	PASS, NO_TEST
14.1.1	8	PASS
14.1.1	9	PASS
14.1.1	10	PASS
14.1.1	11	PASS
14.1.2	12	PASS, NO_TEST_SUPPORT
14.1.3	13	PASS
14.1.4	14	PASS
14.1.4	15	PASS

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**

1877	**Subclause**	**Assertion ID**	**Conforming Results**
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**

1891	**Subclause**	**Assertion ID**	**Conforming Results**
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

14.2.4.2	26	PASS, NO_TEST_SUPPORT
14.2.4.2	27	PASS, NO_TEST_SUPPORT
14.2.4.2	D_3	PASS, NO_OPTION
14.2.4.3	R_1	Reference Assertion, no test results
14.2.4.3	R_3	Reference Assertion, no test results
14.2.4.4	32	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.4.4	35	PASS, NO_OPTION
14.2.4.4	38	PASS, NO_OPTION
14.2.4.4	39	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for timer_gettime

Subclause	Assertion ID	Conforming Results
14.2.4.1	5	PASS[5, 6], NO_OPTION
14.2.4.1	6	PASS[5, 6], NO_OPTION
14.2.4.1	7	PASS, NO_OPTION
14.2.4.1	8	PASS, NO_OPTION
14.2.4.2	timer_gettime	PASS, NO_OPTION
14.2.4.2	25	PASS, NO_OPTION
14.2.4.2	D_4	PASS, NO_OPTION
14.2.4.3	R_2	Reference Assertion, no test results
14.2.4.3	R_4	Reference Assertion, no test results
14.2.4.4	33	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.4.4	36	PASS, NO_OPTION

Assertions for timer_getoverrun

Subclause	Assertion ID	Conforming Results
14.2.4.1	9	PASS[9, 10], NO_OPTION
14.2.4.1	10	PASS[9, 10], NO_OPTION
14.2.4.1	11	PASS, NO_OPTION
14.2.4.1	12	PASS, NO_OPTION
14.2.4.2	timer_getoverrun	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
14.2.4.2	28	PASS, NO_OPTION
14.2.4.2	29	PASS, NO_OPTION, NO_TEST
14.2.4.2	30	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
14.2.4.2	31	PASS, NO_OPTION, NO_TEST
14.2.4.2	D_2	PASS, NO_OPTION
14.2.4.2	D_5	PASS, NO_OPTION
14.2.4.3	R_5	Reference Assertion, no test results
14.2.4.4	34	PASS, NO_OPTION, NO_TEST_SUPPORT
14.2.4.4	37	PASS, NO_OPTION

Assertions for nanosleep

Subclause	Assertion ID	Conforming Results
14.2.5.1	1	PASS[1, 2], NO_OPTION
14.2.5.1	2	PASS[1, 2], NO_OPTION
14.2.5.1	3	PASS, NO_OPTION

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

Assertions for mq_open

Subclause	Assertion ID	Conforming Results
15.2.1.1	1	PASS[1, 2], NO_OPTION
15.2.1.1	2	PASS[1, 2], NO_OPTION
15.2.1.1	3	PASS, NO_OPTION
15.2.1.1	4	PASS, NO_OPTION
15.2.1.2	OpenMaxMqs	PASS[OpenMaxMqs, PCTSOpenMaxMqs]
15.2.1.2	PCTSOpenMaxMqs	PASS[OpenMaxMqs, PCTSOpenMaxMqs]
15.2.1.2	mq_open	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	D_1	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	5	PASS, NO_OPTION
15.2.1.2	6	PASS, NO_OPTION
15.2.1.2	7	PASS, NO_OPTION
15.2.1.2	8	PASS, NO_OPTION
15.2.1.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	D_2	PASS, NO_OPTION
15.2.1.2	D_3	PASS, NO_OPTION
15.2.1.2	R_1	Reference Assertion, no test results
15.2.1.2	10	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	11	PASS
15.2.1.2	12	PASS
15.2.1.2	13	PASS, NO_OPTION
15.2.1.2	14	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	15	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	16	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	17	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	18	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	19	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	20	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	21	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	22	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	23	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	24	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	D_4	PASS, NO_OPTION
15.2.1.2	D_5	PASS, NO_OPTION
15.2.1.2	25	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	R_2	Reference Assertion, no test results
15.2.1.2	R_3	Reference Assertion, no test results
15.2.1.2	26	PASS, NO_OPTION, NO_TEST_SUPPORT, NO_TEST
15.2.1.2	27	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	28	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.2	D_7	PASS, NO_OPTION
15.2.1.3	R_4	Reference Assertion, no test results
15.2.1.3	R_5	Reference Assertion, no test results
15.2.1.4	mq_open_EACCES1	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	mq_open_EACCES2	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.1.4	mq_open_EEXIST	PASS, NO_OPTION, NO_TEST_SUPPORT

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**

2046	**Subclause**	**Assertion ID**	**Conforming Results**
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**

2060	**Subclause**	**Assertion ID**	**Conforming Results**
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

Assertions for mq_send

Subclause	Assertion ID	Conforming Results
15.2.4.1	1	PASS[1, 2], NO_OPTION
15.2.4.1	2	PASS[1, 2], NO_OPTION
15.2.4.1	3	PASS, NO_OPTION
15.2.4.1	4	PASS, NO_OPTION
15.2.4.2	mq_send	PASS, NO_OPTION
15.2.4.2	R_1	Reference Assertion, no test results
15.2.4.2	5	PASS, NO_OPTION
15.2.4.2	6	PASS, NO_OPTION
15.2.4.2	7	PASS, NO_OPTION
15.2.4.2	R_2	Reference Assertion, no test results
15.2.4.2	8	PASS, NO_OPTION
15.2.4.2	9	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.4.2	D_1	PASS, NO_OPTION
15.2.4.3	R_4	Reference Assertion, no test results
15.2.4.3	R_5	Reference Assertion, no test results
15.2.4.4	mq_send_EAGAIN	PASS, NO_OPTION
15.2.4.4	10	PASS, NO_OPTION
15.2.4.4	11	PASS, NO_OPTION
15.2.4.4	mq_send_EINVAL	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.4.4	mq_send_EMSGSIZE	PASS, NO_OPTION
15.2.4.4	12	PASS, NO_OPTION

Assertions for mq_receive

Subclause	Assertion ID	Conforming Results
15.2.5.1	1	PASS[1, 2], NO_OPTION
15.2.5.1	2	PASS[1, 2], NO_OPTION
12.2.5.1	3	PASS, NO_OPTION
12.2.5.1	4	PASS, NO_OPTION
12.2.5.2	mq_receive	PASS, NO_OPTION
12.2.5.2	5	PASS, NO_OPTION
12.2.5.2	6	PASS, NO_OPTION
12.2.5.2	7	PASS, NO_OPTION
12.2.5.2	8	PASS, NO_OPTION, NO_TEST_SUPPORT
12.2.5.2	D_1	PASS, NO_OPTION
12.2.5.2	R_1	Reference Assertion, no test results
12.2.5.2	D_2	PASS, NO_OPTION
12.2.5.3	R_2	Reference Assertion, no test results
12.2.5.3	R_3	Reference Assertion, no test results
12.2.5.4	mq_receive_EAGAIN	PASS, NO_OPTION
12.2.5.4	9	PASS, NO_OPTION
12.2.5.4	10	PASS, NO_OPTION
12.2.5.4	11	PASS, NO_OPTION
12.2.5.4	12	PASS, NO_OPTION
12.2.5.4	13	PASS, NO_OPTION, NO_TEST_SUPPORT

Assertions for mq_notify

Subclause	Assertion ID	Conforming Results
15.2.6.1	1	PASS[1, 2], NO_OPTION
15.2.6.1	2	PASS[1, 2], NO_OPTION
15.2.6.1	3	PASS, NO_OPTION
15.2.6.1	4	PASS, NO_OPTION
15.2.6.2	mq_notify	PASS, NO_OPTION
15.2.6.2	R_1	Reference Assertion, no test results
15.2.6.2	5	PASS, NO_OPTION
15.2.6.2	6	PASS, NO_OPTION
15.2.6.2	7	PASS, NO_OPTION
15.2.6.2	D_1	PASS, NO_OPTION
15.2.6.3	R_2	Reference Assertion, no test results
15.2.6.3	R_3	Reference Assertion, no test results
152.2.6.4	8	PASS, NO_OPTION
15.2.6.4	mq_notify_EBUSY	PASS, NO_OPTION
15.2.6.4	9	PASS, NO_OPTION

Assertions for mq_setattr

Subclause	Assertion ID	Conforming Results
15.2.7.1	1	PASS[1, 2], NO_OPTION
15.2.7.1	2	PASS[1, 2], NO_OPTION
15.2.7.1	3	PASS, NO_OPTION
15.2.7.1	4	PASS, NO_OPTION
15.2.7.2	mq_setattr	PASS, NO_OPTION
15.2.7.2	5	PASS, NO_OPTION
15.2.7.2	D_1	PASS, NO_OPTION
15.2.7.2	6	PASS, NO_OPTION
15.2.7.2	7	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.7.2	D_2	PASS, NO_OPTION
15.2.7.3	R_1	Reference Assertion, no test results
15.2.7.3	R_2	Reference Assertion, no test results
15.2.7.4	8	PASS, NO_OPTION
15.2.7.4	9	PASS, NO_OPTION

Assertions for mq_getattr

Subclause	Assertion ID	Conforming Results
15.2.8.1	1	PASS[1, 2], NO_OPTION
15.2.8.1	2	PASS[1, 2], NO_OPTION
15.2.8.1	3	PASS, NO_OPTION
15.2.8.1	4	PASS, NO_OPTION
15.2.8.2	mq_getattr	PASS, NO_OPTION
15.2.8.2	5	PASS, NO_OPTION, NO_TEST_SUPPORT
15.2.8.2	6	PASS, NO_OPTION
15.2.8.2	7	PASS, NO_OPTION

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

This Annex contains lists of related open systems standards and suggested reading on historical implementations and application programming.

B.1 Related Open Systems Standards

B.1.1. Networking Standards

{B1} ISO 7498: 1984, *Information processing systems – Open Systems Interconnection – Basic Reference Model.* [4)]

{B2} ISO 8072: 1986, *Information processing systems – Open Systems Interconnection – Transport service definition.*

{B3} ISO/IEC 8073: 1988, *Information processing systems – Open Systems Interconnection – Connection oriented transport protocol specification.* [5)]

{B4} ISO 8326: 1987, *Information processing systems – Open Systems Interconnection – Basic connection oriented session service definition.*

{B5} ISO 8327: 1987, *Information processing systems – Open Systems Interconnection – Basic connection oriented session protocol definition.*

{B6} ISO 8348: 1987, *Information processing systems – Data communications – Network service definition.*

{B7} ISO 8473: 1988, *Information processing systems – Data communications – Protocol for providing the connectionless-mode network service.*

{B8} ISO 8571: 1988, *Information processing systems – Open Systems Interconnection – File Transfer, Access and Management.*

{B9} ISO 8649: 1988, *Information processing systems – Open Systems Interconnection – Service definition for the Association Control Service Element.*

{B10} ISO 8650: 1988, *Information processing systems – Open Systems Interconnection – Protocol specification for the Association Control Service Element.*

---

4) ISO documents can be obtained from the ISO office, 1, rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse.

5) IEC documents can be obtained from the IEC office, 3, rue de Varembé, Case Postale 131, CH-1211, Genève 20, Switzerland/Suisse.

{B11} ISO 8802-2: 1989 [IEEE Std 802.2-1989 (ANSI)], *Information processing systems – Local area networks – Part 2: Logical link control.*

{B12} ISO 8802-3: 1989 [IEEE 802.3-1988 (ANSI)], *Information processing systems – Local area networks – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications.*

{B13} ISO/IEC 8802-4: 1990 [IEEE Std 802.4-1990 (ANSI)], *Information technology – Local area networks – Part 4: Token-passing bus access method and physical layer specifications.*

{B14} ISO 8802_5: ... (IEEE 802.5-1989), *Information technology – Local area networks – Part 5: Token ring access method and physical layer specifications.*

{B15} ISO 8822: 1988, *Information processing systems – Open Systems Interconnection – Connection oriented presentation service definition.*

{B16} ISO 8823: 1988, *Information processing systems – Open Systems Interconnection – Connection oriented presentation protocol specification.*

{B17} ISO 8831: 1989, *Information processing systems – Open Systems Interconnection – Job transfer and manipulation concepts and services.*

{B18} ISO 8832: 1989, *Information processing systems – Open Systems Interconnection – Specification of the basic class protocol for job transfer and manipulation.*

{B19} CCITT Recommendation X.25, *Interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCT) for terminals operating in the packet mode and connected to public data networks by dedicated circuit.*[6)]

{B20} CCITT Recommendation X;.212, *Information processing systems – Data communication – Data link service definition for Open Systems Interconnection.*

B.1.2 Language Standards

{B21} ISO 1539: 1980, *Programming languages – FORTRAN.*

{B22} ISO 1989: 1985, *Programming Languages – COBOL.*

{B23} ISO 8652: 1987, *Programming Languages – Ada.*

{B24} ANSI X3.113_1987[7)], *Information systems – Programming language – FULL BASIC.*

{B25} ANSI/IEEE 770X3.97-1983, *Standard Pascal Computer Programming Language.*

{B26} ANSI/MDC X11.1-1984, *Programming Language MUMPS.*

B.1.3 Graphics Standards

{B27} ISO 7942: 1985, *Information processing systems – Computer graphics – Graphical Kernel System (GKS) functional description.*

---

6) CCITT documents can be obtained from the CCITT General Secretariat, International Telecommunications Union, Sales Section, Place des Nations, CH-1211, Genève 20, Switzerland/Suisse.

7) ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY 10018.

{B28} ISO 8632: 1987, *Information processing systems – Computer graphics – Metafile for the storage and transfer of picture description information.*

{B29} ISO/IEC 9592: 1989 (ANSI X3.144-1988), *Information processing systems – Computer graphics – Programmer's hierarchical interactive graphics system (PHIGS).*

B.1.4 Database Standards

{B30} ISO 8907: 1987, *Database Language – NDL.*

{B31} ISO 9075: 1987, *Database Language – SQL.*

B.2 Other Standards

{B32} ISO 639: 1988, *Code for the representation of names of languages.*

{B33} ISO 3166: 1988, *Code for the representation of names of countries.*

{B34} ISO 8859-1: 1987, *Information Processing – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1.*

{B35} ISO 9127: 1988, *Information processing systems – User documentation and cover information for consumer software packages.*

{B36} ISO/IEC 9945-2: ...,[8] *Information technology – Portable operating system interface (POSIX) – Part 2: Shell and utilities.*

{B37} ISO/IEC 10646:...,[9] *Information processing – Multiple octet coded character set.*

{B38} IEEE Std 100-1988, *IEEE Standard Dictionary of Electrical and Electronics Terms.*

{B39} P1003.0/D16,[10] *Draft Guide to the POSIX Open Systems Environment.*

{B40} ISO/IEC TR 10000-1: 1990, *Information technology – Framework and taxonomy of International Standardized Profiles – Part 1: Framework.*

B.3 Historical Documentation and Introductory Texts

{B41} American Telephone and Telegraph Company, *System V Interface Definition (SVID), Issues 2 and 3.* Morristown, NJ: UNIX Press, 1986, 1989.[11]

{B42} American Telephone and Telegraph Company. *UNIX System III Programmer's Manual.* Greensboro, NC: Western Electric Company, October 1981.

{B43} American Telephone and Telegraph Company. *UNIX Time Sharing System: UNIX Programmer's Manual.* 7th ed. Murray Hill, NJ: Bell Laboratories, January 1979.

---

8) To be approved and published.

9) To be approved and published.

10) This unapproved draft document is available from IEEE Publications, 445 Hoes Lane. P.O. Box 1331, Piscataway, NJ 0055-1331. Telephone: 1(800) 678 -IEEE or +1 (908) 981-1393 (outside US).

11) This is one of several documents that represent an industry specification in an area related to POSIX.1. The creators of such documents may be able to identify newer versions that may be interesting.

{B44} "The UNIX System."[12] *AT&T Bell Laboratories Technical Journal*. vol. 63 (8 Part 2), October 1984.

{B45} "UNIX Time-Sharing System."[13] *Bell System Technical Journal*. vol. 57 (6 Part 2), July-August 1978.

{B46} Bach, Maurice J. *The Design of the UNIX Operating System.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

{B47} Dijkstra, E.W. "Solution of a Problem in Concurrent Programming Control," *Communications of the ACM*, vol. 8(9), September 1965, pp. 569-570.

{B48} Furht, Borko, Grostick, Dan, Gluch, David, Rabbat, Guy, Parker, John, and McRoberts, Meg. *Real-Time UNIX Systems: Design and Application Guide*. Boston, MA: Kluwer Academic Publishers, 1991.

{B49} Harbison, Samuel P. and Steele, Guy L. *C: A Reference Manual*. Englewood Cliffs, NJ: Prentice_Hall, 1987.

{B50} Kernighan, Brian W. and Ritchie, Dennis M. *The C Programming Language*. Englewood Cliffs, NJ: Prentice-Hall, 1978.

{B51} Kernighan, Brian W. and Pike, Rob. *The UNIX Programming Environment*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

{B52} Leffler, Samuel J., McKusick, Marshall Kirk, Karels, Michael J. , Quarterman, John S., and Stettner, Armando. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Reading, MA: Addison-Wesley, 1988.

{B53} McGilton, Henry and Morgan, Rachel. *Introducing the UNIX System.* New York: McGraw-Hill (BYTE BOOKS), 1983.

{B54} Organick, Elliot I. *The Multics System: An Examination of Its Structure.* Cambridge, MA: the MIT Press, 1972.

{B55} Quarterman, John S., Silberschatz, Abraham, and Peterson, James L. "4.2BSD and 4.3BSD as Examples of the UNIX System." *ACM Computing Surveys*. vol. 17(4), December 1985, pp. 379-418.

{B56} Ritchie, Dennis M. "Reflections on Software Research." *Communiciations of the ACM, vol. 27(8),* August 1984, pp. 758-760. ACM Turing Award Lecture.

{B57} Ritchie, Dennis. "The Evolution of the UNIX Time-Sharing System." *AT&T Bell Laboratories Technical Journal*. vol. 63(8), October 1984, pp. 1577-1593.

{B58} Ritchie, D.M. and Thompson, K. "The UNIX Time-Sharing System." *Communications of the ACM*. vol.7(7), July 1974, pp. 365-375. This is the original paper, which describes Version 6.

{B59} Ritchie, D.M. and Thompson, K. "The UNIX Time-Sharing System." *Bell System Technical Journal*. vol. 57 (6 Part 2), July-August 1978, pp. 1905-1929. This is a revised version and describes Version 7.

{B60} Ritchie, Dennis M. "Unix: A Dialectic." *Winter 1987 USENIX Association Conference Proceedings, Washington, D.C.*, pp. 29-34. Berkeley, CA: USENIX Association, January 1987.

{B61} Rochkind, Marc J. *Advanced UNIX Programming.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

{B62} University of California at Berkeley – Computer Science Research Group. *4.3 Berkeley Software Distribution, Virtual VAX-11 Version.* Berkeley, CA: The Regents of the University of California, April 1986.

---

[12] This entire edition is devoted to the UNIX system.

[13] This entire edition is devoted to the UNIX time-sharing system.

{B63} /usr/group Standards Committee. *1984 /usr/group Standard.* Santa Clara, CA: Uni_Forum, 1984.

{B64} X/Open Company, Ltd. *X/Open Portability Guide, Issue 2.* Amsterdam: Elsevier Science Publishers, 1987.

{B65} X/Open Company, Ltd. *X/Open Portability Guide, Issue 3.* Englewood Cliffs, NJ: Prentice-Hall, 1989.

B.4 Other Sources of Information

{B66} ISO/IEC JTC 1 N1335, *Final Report of ISO/IEC JTC 1 TSG-1 on Standards Necessary to Define Interfaces for Application Portability (IAP).*

This page is intentionally blank.

Identifier Index

Alphabetic Topical Index