

IEEE Standard for Information Technology—Portable Operating System Interface (POSIX[®])—Part 1: System Application Program Interface (API)— Amendment d: Additional Realtime Extensions [C Language]

Sponsor

Portable Application Standards Committee
of the
IEEE Computer Society

Approved 16 September 1999

IEEE-SA Standards Board

Abstract: This standard is part of the POSIX series of standards for applications and user interfaces to open systems. It defines the applications interface to system services for spawning a process, timeouts for blocking services, sporadic server scheduling, execution time clocks and timers, and advisory information for file management. This standard is stated in terms of its C binding.

Keywords: API, application portability, C (programming language), data processing, open systems, operating system, portable application, POSIX, realtime

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

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

Copyright © 2000 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 16 June 2000. Printed in the United States of America.

Print: ISBN 0-7381-1815-X SH94790
PDF: ISBN 0-7381-1816-8 SS94790

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

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

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

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

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

Comments on standards and requests for interpretations should be addressed to:

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

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

IEEE is the sole entity that may authorize the use of certification marks, trademarks, or other designations to indicate compliance with the materials set forth herein.

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

Contents

	PAGE
Introduction	v
Section 1: General	1
1.1 Scope	1
1.3 Conformance	2
Section 2: Terminology and General Requirements	5
2.2 Definitions	5
2.3 General Concepts	6
2.7 C Language Definitions	7
2.8 Numerical Limits	8
2.9 Symbolic Constants	10
Section 3: Process Primitives	13
3.1 Process Creation and Execution	13
3.1.1 Process Creation	13
3.1.2 Execute a File	13
3.1.4 Spawn File Actions	14
3.1.5 Spawn Attributes	16
3.1.6 Spawn a Process	20
3.2 Process Termination	25
3.2.1 Wait for Process Termination	25
Section 4: Process Environment	27
4.8 Configurable System Variables	27
4.8.1 Get Configurable System Variables	27
Section 5: Files and Directories	29
5.7 Configurable Pathname Variables	29
5.7.1 Get Configurable Pathname Variables	29
Section 6: Input and Output Primitives	31
6.7 Asynchronous Input and Output	31
6.7.1 Data Definitions for Asynchronous Input and Output	31
Section 11: Synchronization	33
11.2 Semaphore Functions	33
11.2.6 Lock a Semaphore	33
11.2.7 Unlock a Semaphore	35
11.3 Mutexes	35
11.3.3 Locking and Unlocking a Mutex	35
Section 13: Execution Scheduling	39
13.1 Scheduling Parameters	39

13.2	Scheduling Policies	39
13.2.3	SCHED_OTHER	39
13.2.4	SCHED_SPORADIC	40
13.3	Process Scheduling Functions	41
13.3.1	Set Scheduling Parameters	41
13.3.3	Set Scheduling Policy and Scheduling Parameters	42
13.4	Thread Scheduling	43
13.4.1	Thread Scheduling Attributes	43
13.4.3	Scheduling Allocation Domain	43
13.4.4	Scheduling Documentation	44
13.5	Thread Scheduling Functions	44
13.5.1	Thread Creation Scheduling Attributes	44
13.5.2	Dynamic Thread Scheduling Parameters Access	45
Section 14:	Clocks and Timers	47
14.2	Clock and Timer Functions	47
14.2.1	Clocks	47
14.2.2	Create a Per-Process Timer	48
14.3	Execution Time Monitoring	48
14.3.1	CPU-time Clock Characteristics	48
14.3.2	Accessing a Process CPU-time Clock	49
14.3.3	Accessing a Thread CPU-time Clock	50
Section 15:	Message Passing	53
15.2	Message Passing Functions	53
15.2.4	Send a Message to a Message Queue	53
15.2.5	Receive a Message from a Message Queue	55
Section 16:	Thread Management	59
16.2	Thread Functions	59
16.2.2	Thread Creation	59
Section 18:	Thread Cancellation	61
18.1	Thread Cancellation Overview	61
18.1.2	Cancellation Points	61
Section 19:	Advisory Information	63
19.1	I/O Advisory Information and Space Control	63
19.1.1	File Advisory Information	63
19.1.2	File Space Control	64
19.2	Memory Advisory Information and Alignment Control	66
19.2.1	Memory Advisory Information	66
19.2.2	Aligned Memory Allocation	67
Annex A (informative)	Bibliography	69
A.2	Other Standards	69
A.3	Historical Documentation and Introductory Texts	69
Annex B (informative)	Rationale and Notes	71
B.2	Definitions and General Requirements	71
B.3	Process Primitives	71
B.13	Execution Scheduling	87

B.14 Clocks and Timers	91
B.19 Advisory Information	102
Identifier Index	105
Alphabetic Topical Index	107

FIGURES

Figure B-1 – <i>posix_spawn()</i> Equivalent	86
Figure B-2 – I/O Redirection with <i>posix_spawn()</i>	86
Figure B-3 – Spawning a new Userid Process	87
Figure B-4 – Spinlock Implementation	97
Figure B-5 – Condition Wait Implementation	98
Figure B-6 – <i>pthread_join()</i> with timeout	101

TABLES

Table 2-3a – Optional Minimum Values	8
Table 2-5a – Optional Run-Time Invariant Values	9
Table 2-6a – Optional Pathname Variable Values	10
Table 2-10a – Versioned Compile-Time Symbolic Constants	11
Table 4-3 – Optional Configurable System Variables	27
Table 5-3 – Optional Configurable Pathname Variables	29

Introduction

(This introduction is not a normative part of IEEE Std 1003.1d-1999, Information Technology—Portable Operating System Interface (POSIX®)—Part 1: System Application Program Interface (API)—Amendment d: Additional Realtime Extensions [C Language])

Editor's Note: This introduction consists of material that will eventually be integrated into the base POSIX.1¹⁾ standard's introduction (and the portion of Annex B that contains general rationale about the standard). The introduction contains text that was previously held in either the foreword or scope. As this portion of the standard is for information only, specific details of the integration with POSIX.1 are left as an editorial exercise. The section and subclause structure of this document follows that of POSIX.1. Sections that are not amended by this standard are omitted.

The purpose of this document is to supplement the base standard with interfaces and functionality for applications having realtime requirements.

This standard defines systems interfaces to support the source portability of applications with realtime requirements. The system interfaces are all extensions of or additions to ISO/IEC 9945-1: 1990, *Portable Operating System Interface for Computer Environments*, as amended by POSIX.1b and POSIX.1c. Although rooted in the culture defined by ISO/IEC 9945-1: 1990, the interfaces are focused upon the realtime application requirements, which were beyond the ISO/IEC 9945-1: 1990 scope. The interfaces included in this standard are additions to the set required to make ISO/IEC 9945-1: 1990 minimally usable to realtime applications on single processor systems.

The definition of *realtime* used in defining the scope of this standard is

Realtime in operating systems: the ability of the operating system to provide a required level of service in a bounded response time.

The key elements of defining the scope are

- (1) defining a sufficient set of functionality to cover the realtime application program domain in the areas not covered by POSIX.1b and POSIX.1c;
- (2) defining sufficient performance constraints and performance-related functions to allow a realtime application to achieve deterministic response from the system; and
- (3) specifying changes or additions to improve or complete the definition of the facilities specified in the previous real-time or threads extensions covered by POSIX.1b and POSIX.1c.

Wherever possible, the requirements of other application environments were included in the interface definition. The specific areas are noted in the scope overviews of each of the interface areas given below.

The specific functional areas included in this standard and their scope include

- Spawn a process: new system services to spawn the execution of a new process in an efficient manner.
- Timeouts for some blocking services: additional services that provide a timeout capability to system services already defined in POSIX.1b and POSIX.1c, thus

1) See 2.3.3 in this standard for more information about these references.

- 39 allowing the application to include better error detection and recovery capabilities.
- 40 — Sporadic server scheduling: the addition of a new scheduling policy appropriate for
- 41 scheduling aperiodic processes or threads in hard realtime applications.
- 42 — Execution time clocks and timers: the addition of new clocks that measure the exe-
- 43 cution times of processes or threads, and the possibility to create timers based upon
- 44 these clocks, for runtime detection (and treatment) of execution time overruns.
- 45 — Advisory information for file management: addition of services that allow the appli-
- 46 cation to specify advisory information that can be used by the system to achieve
- 47 better or even deterministic response times in file management or input and output
- 48 (I/O) operations.

49 There are two other functional areas that were included in the scope of this standard, but

50 the balloting group considered that they were not ready yet for standardization:

- 51 — Device control: a new service to pass control information and commands between
- 52 the application and device drivers.
- 53 — Interrupt control: new services that allow the application to directly handle
- 54 hardware interrupts.

55 This standard has been defined exclusively at the source code level for the C programming

56 language. Although the interfaces will be portable, some of the parameters used by an

57 implementation may have hardware or configuration dependencies.

58 **Related Standards Activities**

59 Activities to extend this standard to address additional requirements are in progress, and

60 similar efforts can be anticipated in the future.

61 The following areas are under active consideration at this time or are expected to become

62 active in the near future:²⁾

- 63 (1) Additional system application program interfaces (APIs) in C language
- 64 (2) Ada and FORTRAN language bindings to (1)
- 65 (3) Shell and utility facilities
- 66 (4) Verification testing methods
- 67 (5) Realtime facilities
- 68 (6) Tracing facilities
- 69 (7) Fault tolerance
- 70 (8) Checkpoint/restart facilities
- 71 (9) Resource limiting facilities
- 72 (10) Network interface facilities
- 73 (11) System administration

74

75 2) A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from

76 the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903; Telephone:

77 +1 202 371-0101; FAX: +1 202 728-9614. Working drafts of POSIX standards under development are available

78 from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ

79 08855-1331 (<http://www.standards.ieee.org/>).

80 (12) Profiles describing application- or user-specific combinations of Open Systems
81 standards

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

83 Extensions are approved as “amendments” or “revisions” to this document, following the
84 IEEE and ISO/IEC procedures.

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

87 If you have interest in participating in the Portable Application Standards Committee
88 (PASC) working groups addressing these issues, please send your name, address, and
89 phone number to

90 Secretary, IEEE Standards Board
91 Institute of Electrical and Electronics Engineers, Inc.
92 P.O. Box 1331
93 445 Hoes Lane
94 Piscataway, NJ 08855-1331
95 USA

96 When writing, ask to have your letter forwarded to the chairperson of the appropriate
97 PASC working group. If you have interest in participating in this work at the interna-
98 tional level, contact your International Organization for Standardization/International
99 Electrotechnical Committee (ISO/IEC) national body.

100 This standard was prepared by the system services working group—realtime, sponsored
101 by the Portable Application Standards Committee of the IEEE Computer Society. At the
102 time this standard was approved, the membership of the system services working group—
103 realtime was as follows:

104 **Portable Application Standards Committee**

105 Chair: Lowell Johnson
106 Vice Chair: Joseph M. Gwinn
107 Functional Vice Chairs: Jay Ashford
108 Andrew Josey
109 Curtis Royster Jr.
110 Secretary: Nick Stoughton

111 **IEEE System Services Working Group—Realtime**

112 Chair: Joseph M. Gwinn
113 Susan Corwin (to 1995)
114 Secretary: Karen D. Gordon
115 Franklin C. Prindle (1996)
116 Lee Schermerhorn (to 1995)
117 Editor: Michael González
118 Robert D. Luken (to 1997)
119 Technical reviewers: Steve Brosky
120 Peter Dibble
121 Christoph Eck
122 Michael González
123 Karen D. Gordon
124 Joseph M. Gwinn
125 Franklin C. Prindle
126 Ballot coordinators: James T. Oblinger
127 Duane Hughes (to 1996)

128 **Working Group**

129 Ray Alderman	Bill Gallmeister	Kent Long
130 Larry Anderson	Michael González	Robert D. Luken
131 Pierre-Jean Arcos	Karen D. Gordon	James T. Oblinger
132 Charles R. Arnold	Randy Greene	Offer Pazy
133 V. Raj Avula	Rick Greer	Franklin C. Prindle
134 Theodore P. Baker	Joseph M. Gwinn	François Riche
135 Robert Bamed	Steven A. Haaser	Gordon W. Ross
136 Richard M. Bergman	Barbara Haleen	Curtis Royster, Jr.
137 Nawaf Bitar	Geoffrey R. Hall	Webb Scales
138 Steve Brosky	Patrick Hebert	Lee Schermerhorn
139 David Butenhof	Mary R. Hermann	Lui Sha
140 Hans Petter Christiansen	David Hughes	Del Swanson
141 Susan Corwin	Duane Hughes	Barry Traylor
142 Bill Cox	Michael B. Jones	Stephen R. Wali
143 Peter Dibble	Steve Kleiman	Andrew E. Wheeler, Jr.
144 Christoph Eck	Robert Knighten	David Wilner
145 Michael Feustel	C. Douglass Locke	John Zolnowsky

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

147	Phillip R. Acuff	Michael González	Diane Paul
148	Alejandro Alonso-Muñoz	Karen D. Gordon	Charles Pfleeger
149	Pierre-Jean Arcos	Mars J. Gralia	John Pijanowski
150	Jay Ashford	Joseph M. Gwinn	Franklin C. Prindle
151	Theodore P. Baker	Steven A. Haaser	Juan Antonio de la Puente
152	Robert Barned	Chris J. Harding	François Riche
153	Barbara K. Beauchamp	Barry Hedquist	Chuck Roark
154	Keith Bierman	Karl Heubaum	Hyman Rosen
155	Nawaf Bitar	Andrew R. Huber	Helmut Roth
156	David Black	Duane Hughes	Curtis Royster
157	David J. Blackwood	Petr Janecek	Richard Scalzo
158	Shirley Bockstahler-Brandt	Lowell G. Johnson	Richard Seibel
159	Mark Brown	Michael B. Jones	Keith Shillington
160	Alan Burns	Andrew Josey	W. Olin Sibert
161	Gregory Bussiere	Michael J. Karels	Jacob Slonim
162	H. L. Catala	James J. Keys	Nicholas M. Stoughton
163	Andrew B. Cheese	Martin J. Kirk	Gregory Swain
164	Michael W. Condry	Thomas M. Kurihara	Efstathios D. Sykas
165	Donald Cragun	Mark Larsen	Donn S. Terry
166	John S. Davies	Martin Leisner	Mark-Rene Uchida
167	Richard P. Draves	Bruce Lewis	Michael W. Vannier
168	Christoph Eck	C. Douglass Locke	Charlotte Wales
169	Philip H. Enslow	Roger J. Martin	Frederick N. Webb
170	W. Douglas Findley, Jr.	Finnbarr P. Murphy	Laurence Wolfe
171	Bill Gallmeister	Richard E. Neese	Oren Yuen
172	Michel P. Gien	James T. Oblinger	Ming De Zhou

173 The following organizational representatives voted on this standard:

174	James T. Oblinger	Diane Paul	Andrew Josey
175	NGCR OSSWG	SAE	X/Open Co. Ltd.

176 When the IEEE-SA Standards Board approved this standard on 16 September
177 1999, it had the following membership:

178 **Richard J. Holleman**, *Chair*
179 **Donald N. Heirman**, *Vice Chair*
180 **Judith Gorman**, *Secretary*

181	Satish K. Aggarwal	James H. Gurney	Louis-François Pau
182	Dennis Bodson	Lowell G. Johnson	Ronald C. Petersen
183	Mark D. Bowman	Robert Kennelly	Gerald H. Peterson
184	James T. Carlo	E.G. "Al" Kiener	John B. Posey
185	Gary R. Engmann	Joseph L. Koepfinger*	Gary S. Robinson
186	Harold E. Epstein	L. Bruce McClung	Akio Tojo
187	Jay Forster*	Daleep C. Mohla	Hans E. Weinrich
188	Ruben D. Garzon	Robert F. Munzner	Donald W. Zispe
189	*Member emeritus		

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

191 Robert E. Hebner

192 Yvette Ho Sang

193 *IEEE Standards Project Editor*

Information Technology—Portable Operating System Interface (POSIX®)—Part 1: System Application Program Interface (API)— Amendment d: Additional Realtime Extensions [C Language]

Section 1: General

1.1 Scope

This standard defines realtime extensions to a standard operating system interface and environment to support application portability at the source-code level. It is intended to be used by both application developers and system implementers.

This standard will not change the base standard that it amends (including any existing amendments) in such a way as to cause implementations or strictly conforming applications to no longer conform.

The scope is to take existing realtime operating system practice and add it to the base standard. The definition of *realtime* used in defining the scope of this standard is

Realtime in operating systems: the ability of the operating system to provide a required level of service in a bounded response time.

The key elements of defining the scope are

- (1) defining a sufficient set of functionality to cover a significant part of the realtime application programming domain, and
- (2) defining sufficient performance constraints and performance related functions to allow a realtime application to achieve deterministic response from the system.

Specifically within the scope is to define interfaces that do not preclude high performance implementations on traditional uniprocessor realtime systems. Wherever possible, the requirements of other application environments were included in the interface definition. The specific functional areas included in this document and their scope include

- Spawn: A process creation primitive useful for systems that have difficulty with `fork()` and as an efficient replacement for `fork()/exec`.
- Timeouts: Alternatives to blocking primitives that provide a timeout parameter to be specified.
- Execution time monitoring: A set of execution time monitoring primitives that allow on-line measuring of thread and process execution times.
- Sporadic server: A scheduling policy for threads and processes that reserves a certain amount of execution capacity for processing aperiodic events at a given priority level.
- Advisory information: An interface that advises the implementation on (portable) application behavior so that it can optimize the system.

Two other functional areas were included in the scope of this standard, but the balloting group considered that they were not ready yet for standardization

- Device control: A portable application interface to nonportable special devices.
- Interrupt control: An interface that allows a process or thread to capture an interrupt, to block awaiting the arrival of an interrupt, and to protect critical sections of code that are contended for by a user-written interrupt service routine.

This standard has been defined exclusively at the source code level. Additionally, although the interfaces will be portable, some of the numeric parameters used by an implementation may have hardware dependencies.

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.3 Conforming Implementation Options

⇒ **1.3.1.3 Conforming Implementation Options** *Add the following to the table of implementation options that warrant requirement by applications or in specifications:*

52	{_POSIX_ADVISORY_INFO}	Advisory Information option (in 2.9.3)
53	{_POSIX_CPUTIME}	Process CPU-Time Clocks option (in 2.9.3)
54	{_POSIX_SPAWN}	Spawn option (in 2.9.3)
55	{_POSIX_SPORADIC_SERVER}	Process Sporadic Server option (in 2.9.3)
56	{_POSIX_THREAD_CPUTIME}	Thread CPU-Time Clocks option (in 2.9.3)
57	{_POSIX_THREAD_SPORADIC_SERVER}	Thread Sporadic Server option (in 2.9.3)
58	{_POSIX_TIMEOUTS}	Timeouts option (in 2.9.3)

1.3 Conformance

Section 2: Terminology and General Requirements

2.2 Definitions

2.2.2 General Terms

⇒ **2.2.2 General Terms** *Modify the contents of 2.2.2 to add the following definitions in the correct sorted order (disregarding the subclause numbers shown here).*

2.2.2.1 CPU time [execution time]: The time spent executing a process or thread, including the time spent executing system services on behalf of that process or thread. If the Threads option is supported, then the value of the CPU-time clock for a process is implementation defined. With this definition the sum of all the execution times of all the threads in a process might not equal the process execution time, even in a single-threaded process, because implementations may differ in how they account for time during context switches or for other reasons.

2.2.2.2 CPU-time clock: A clock that measures the execution time of a particular process or thread.

2.2.2.3 CPU-time timer: A timer attached to a CPU-time clock.

2.2.2.4 execution time: See *CPU time* in 2.2.2.1.

2.2.3 Abbreviations

For this standard, the following abbreviations apply:

2.2.3.1 C Standard: ISO/IEC 9899: 1995, *Information technology—Programming languages—C*.

2.2.3.2 POSIX.1: ISO/IEC 9945-1: 1996, (IEEE Std 1003.1-1996), *Information Technology—Portable Operating System Interface (POSIX®)—Part 1: System Application Program Interface (API) [C Language]*.

24 **2.2.3.3 POSIX.1b:** IEEE Std 1003.1b-1993, *Information Technology—Portable*
 25 *Operating System Interface (POSIX®)—Part 1: System Application Program*
 26 *Interface (API)—Amendment b: Realtime Extensions [C Language]*, as amended
 27 *by IEEE Std 1003.1i-1995, Information Technology—Portable Operating System*
 28 *Interface (POSIX®)—Part 1: System Application Program Interface (API)—*
 29 *Amendment i: Technical Corrigenda to Realtime Extension [C Language].*

30 **2.2.3.4 POSIX.1c:** IEEE Std 1003.1c-1995, *Information Technology—Portable*
 31 *Operating System Interface (POSIX®)—Part 1: System Application Program*
 32 *Interface (API)—Amendment c: Threads Extension [C Language].*

33 **2.2.3.5 POSIX.1d:** IEEE Std 1003.1d-1999, *this standard.*

34 **2.2.3.6 POSIX.5** ISO/IEC 14519:1998 {B1}¹⁾, *POSIX® Ada Language Interfaces—*
 35 *Binding for System Application Program Interfaces (API) including Realtime*
 36 *Extensions.* (This standard includes IEEE Std 1003.5-1992 and IEEE Std 1003.5b-
 37 1996.)

38 **2.3 General Concepts**

39 ⇒ **2.3 General Concepts—measurement of execution time:** *Add the follow-*
 40 *ing subclause, in the proper order, to the existing items in 2.3:*

41 **2.3.1 measurement of execution time:** The mechanism used to measure exe-
 42 *cution time shall be implementation defined. The implementation shall also*
 43 *define to whom will be charged the CPU time that is consumed by interrupt*
 44 *handlers and system services on behalf of the operating system. Execution or CPU*
 45 *time is defined in 2.2.2.1.*

46 _____
 47 1) The numbers in curly brackets, when preceded by a “B”, correspond to the numbers of the
 48 bibliography in Annex A.

2.7 C Language Definitions

2.7.3 Headers and Function Prototypes

⇒ **2.7.3 Headers and Function Prototypes** *Add the following text after the sentence “For other functions in this part of ISO/IEC 9945, the prototypes or declarations shall appear in the headers listed below.”:*

Presence of some prototypes or declarations is dependent on implementation options. Where an implementation option is not supported, the prototype or declaration need not be found in the header.

⇒ **2.7.3 Headers and Function Prototypes** *Modify the contents of subclause 2.7.3 to add the following optional headers and functions, at the end of the current list of headers and functions.*

If the Advisory Information option is supported:

<fcntl.h> *posix_fadvise(), posix_madvise(), posix_fallocate()*

If the Message Passing option and the Timeouts option are supported:

<mqueue.h> *mq_timedsend(), mq_timedreceive()*

If the Thread CPU-Time Clocks option is supported:

<pthread.h> *pthread_getcpuclockid()*

If the Threads option and the Timeouts option are supported:

<pthread.h> *pthread_mutex_timedlock()*

If the Semaphores option and the Timeouts option are supported:

<semaphore.h> *sem_timedwait()*

If the Spawn option is supported:

<spawn.h> *posix_spawn(), posix_spawnnp(),
 posix_spawn_file_actions_init(),
 posix_spawn_file_actions_destroy(),
 posix_spawn_file_actions_addclose(),
 posix_spawn_file_actions_adddup2(),
 posix_spawn_file_actions_addopen(),
 posix_spawnattr_init(), posix_spawnattr_destroy(),
 posix_spawnattr_getflags(), posix_spawnattr_setflags(),
 posix_spawnattr_getpgroup(),
 posix_spawnattr_setpgroup(),
 posix_spawnattr_getsigmask(),
 posix_spawnattr_setsigmask(),
 posix_spawnattr_getsigdefault(),
 posix_spawnattr_setsigdefault()*

If the Spawn option and the Process Scheduling option are supported:

86

<spawn.h>

posix_spawnattr_getschedpolicy(),

87

posix_spawnattr_setschedpolicy(),

88

posix_spawnattr_getschedparam(),

89

posix_spawnattr_setschedparam()

90 If the Advisory Information option is supported:

91

<stdlib.h>

posix_memalign()

92 If the Process CPU-Time Clocks option is supported:

93

<time.h>

clock_getcpuclockid()

94 **2.8 Numerical Limits**

95 **2.8.2 Minimum Values**

96

⇒ **2.8.2 Minimum Values**

97

Add the following text after the sentence starting “The symbols in Table 2-3 shall be defined in...”

98 The symbols in Table 2-3a shall be defined in **<limits.h>** with the values
 99 shown if the associated option is supported.

100 ⇒ **2.8.2 Minimum Values** *Add Table 2-3a, described below, after Table 2-3.*

101 **Table 2-3a – Optional Minimum Values**

102

Name	Description	Value	Option
103 {_POSIX_SS_REPL_MAX}	104 The number of replenishment 105 operations that may be 106 simultaneously pending for 107 a particular sporadic server scheduler.	4	Process Sporadic Server or Thread Sporadic Server

108 **2.8.4 Run-Time Invariant Values (Possibly Indeterminate)**

109

⇒ **2.8.4 Run-Time Invariant Values (Possibly Indeterminate)**

110

Replace the whole subclause by the following text:

111 The symbols that appear in Table 2-5 that have determinate values shall be
 112 defined in **<limits.h>**. The symbols that appear in Table 2-5a that have
 113 determinate values shall be defined in **<limits.h>** if the associated option is
 114 supported. If any of the values in Table 2-5 or Table 2-5a has a value that is
 115 greater than or equal to the stated minimum, but is indeterminate, a definition
 116 for that value shall not be defined in **<limits.h>**.

117 This indetermination might depend on the amount of available memory space
 118 on a specific instance of a specific implementation. For the values defined in

119 Table 2-5, the actual value supported by a specific instance shall be provided by
120 the *sysconf()* function. For the values defined in Table 2-5a, the actual value
121 supported by a specific instance shall be provided by the *sysconf()* function if
122 the associated option is supported.

123 ⇒ **2.8.4 Run-Time Invariant Values (Possibly Indeterminate)** *Add*
124 Table 2-5a, described next, after Table 2-5.

125 **Table 2-5a – Optional Run-Time Invariant Values**
126 **(Possibly Indeterminate)**

127 Name	Description	Minimum Value	Option
128 {SS_REPL_MAX}	129 The maximum number of 130 replenishment opera- 131 tions that may be 132 simultaneously pending 133 for a particular 134 sporadic server scheduler.	{_POSIX_SS- REPL_MAX}	Process Sporadic Server or Thread Sporadic Server

135 **2.8.5 Pathname Variable Values**

136 ⇒ **2.8.5 Pathname Variable Values** *Replace the reference to Table 2-6 in the*
137 *first paragraph of this subclause by:*

138 Table 2-6 or Table 2-6a

139 ⇒ **2.8.5 Pathname Variable Values** *Replace the sentence “The actual value*
140 *supported for a specific pathname shall be provided by the pathconf() function”*
141 *with the following text:*

142 For the values defined in Table 2-6, the actual value supported for a specific
143 pathname shall be provided by the *pathconf()* function. For the values defined
144 in Table 2-6a, the actual value supported for a specific pathname shall be pro-
145 vided by the *pathconf()* function if the associated option is supported.

146 ⇒ **2.8.5 Pathname Variable Values** *Add Table 2-6a, described next, after*
147 *Table 2-6.*

Table 2-6a – Optional Pathname Variable Values

Name	Description	Minimum Values	Option
{POSIX_REC_INCR_XFER_SIZE}	Recommended increment for file transfer sizes between the {POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.	<i>not specified</i>	Advisory Information
{POSIX_ALLOC_SIZE_MIN}	Minimum number of bytes of storage actually allocated for any portion of a file.	<i>not specified</i>	Advisory Information
{POSIX_REC_MAX_XFER_SIZE}	Maximum recommended file transfer size.	<i>not specified</i>	Advisory Information
{POSIX_REC_MIN_XFER_SIZE}	Minimum recommended file transfer size.	<i>not specified</i>	Advisory Information
{POSIX_REC_XFER_ALIGN}	Recommended file transfer buffer alignment.	<i>not specified</i>	Advisory Information

2.9 Symbolic Constants

2.9.3 Compile-Time Symbolic Constants for Portability Specifications

⇒ 2.9.3 Compile-Time Symbolic Constants for Portability Specifications
 Change the first words in the first paragraph, currently saying “The constants in Table 2-10 may be used...” to the following:

The constants in Table 2-10 and Table 2-10a may be used...

⇒ 2.9.3 Compile-Time Symbolic Constants for Portability Specifications
 Add the following sentence at the end of the first paragraph:

If any of the constants in Table 2-10a is defined, it shall be defined with the value shown in that table. This value represents the version of the associated option that is supported by the implementation.

⇒ 2.9.3 Compile-Time Symbolic Constants for Portability Specifications
 Add Table 2-10a, shown below, after Table 2-10.

179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201

Table 2-10a – Versioned Compile-Time Symbolic Constants

Name	Value	Description
{_POSIX_ADVISORY_INFO}	199909L	If this symbol is defined, the implementation supports the Advisory Information option.
{_POSIX_CPUTIME}	199909L	If this symbol is defined, the implementation supports the Process CPU-Time Clocks option.
{_POSIX_SPAWN}	199909L	If this symbol is defined, the implementation supports the Spawn option.
{_POSIX_SPORADIC_SERVER}	199909L	If this symbol is defined, the implementation supports the Process Sporadic Server option.
{_POSIX_THREAD_CPUTIME}	199909L	If this symbol is defined, the implementation supports the Thread CPU-Time Clocks option.
{_POSIX_THREAD_SPORADIC_SERVER}	199909L	If this symbol is defined, the implementation supports the Thread Sporadic Server option.
{_POSIX_TIMEOUTS}	199909L	If this symbol is defined, the implementation supports the Timeouts option.

202 ⇒ **2.9.3 Compile-Time Symbolic Constants for Portability Specifications**
203 *Add the following paragraphs before the last paragraph in 2.9.3:*

204 If the symbol {_POSIX_SPORADIC_SERVER} is defined, then the symbol
205 {_POSIX_PRIORITY_SCHEDULING} shall also be defined. If the symbol
206 {_POSIX_THREAD_SPORADIC_SERVER} is defined, then the symbol {_POSIX-
207 THREAD_PRIORITY_SCHEDULING} shall also be defined.

208 If the symbol {_POSIX_CPUTIME} is defined, then the symbol {_POSIX_TIMERS}
209 shall also be defined. If the symbol {_POSIX_THREAD_CPUTIME} is defined,
210 then the symbol {_POSIX_TIMERS} shall also be defined.

Section 3: Process Primitives

1 **3.1 Process Creation and Execution**

2 **3.1.1 Process Creation**

3 **3.1.1.2 Description**

4 **⇒ 3.1.1.2 Process Creation—Description** *Add the following paragraphs to the*
 5 *description of the fork() function:*

6 If {_POSIX_CPUTIME} is defined:

7 The initial value of the CPU-time clock of the child process shall be set to
 8 zero.

9 If {_POSIX_THREAD_CPUTIME} is defined:

10 The initial value of the CPU-time clock of the single thread of the child
 11 process shall be set to zero.

12 **3.1.2 Execute a File**

13 **3.1.2.2 Description**

14 **⇒ 3.1.2.2 Execute a File—Description** *Add the following paragraph to the*
 15 *description of the family of exec functions.*

16 If {_POSIX_CPUTIME} is defined:

17 The new process image shall inherit the CPU-time clock of the calling
 18 process image. This inheritance means that the process CPU-time clock
 19 of the process being *execed* shall not be reinitialized or altered as a
 20 result of the *exec* function other than to reflect the time spent by the
 21 process executing the *exec* function itself.

22 If {_POSIX_THREAD_CPUTIME} is defined:

23 The initial value of the CPU-time clock of the initial thread of the new
 24 process image shall be set to zero.

25 ⇒ **3.1 Process Creation and Execution** Add the following subclauses:

26 **3.1.4 Spawn File Actions**

27 Functions: *posix_spawn_file_actions_init()*, *posix_spawn_file_actions_destroy()*,
 28 *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*,
 29 *posix_spawn_file_actions_addopen()*.

30 **3.1.4.1 Synopsis**

```
31 #include <sys/types.h>
32 #include <spawn.h>

33 int posix_spawn_file_actions_init(
34     posix_spawn_file_actions_t *file_actions);

35 int posix_spawn_file_actions_destroy(
36     posix_spawn_file_actions_t *file_actions);

37 int posix_spawn_file_actions_addclose(
38     posix_spawn_file_actions_t *file_actions,
39     int fildes);

40 int posix_spawn_file_actions_adddup2(
41     posix_spawn_file_actions_t *file_actions,
42     int fildes, int newfildes);

43 int posix_spawn_file_actions_addopen(
44     posix_spawn_file_actions_t *file_actions,
45     int fildes, const char *path,
46     int oflag, mode_t mode);
```

47 **3.1.4.2 Description**

48 If `{_POSIX_SPAWN}` is defined:

49 A spawn file actions object is of type *posix_spawn_file_actions_t* (defined in
 50 `<spawn.h>`) and is used to specify a series of actions to be performed by a
 51 *posix_spawn()* or *posix_spawnnp()* operation in order to arrive at the set of
 52 open file descriptors for the child process given the set of open file descrip-
 53 tors of the parent. This standard does not define comparison or assignment
 54 operators for the type *posix_spawn_file_actions_t*.

55 The *posix_spawn_file_actions_init()* function initializes the object refer-
 56 enced by *file_actions* to contain no file actions for *posix_spawn()* or
 57 *posix_spawnnp()* to perform.

58 The effect of initializing an already initialized spawn file actions object is
 59 undefined.

60 The *posix_spawn_file_actions_destroy()* function destroys the object refer-
 61 enced by *file_actions*; the object becomes, in effect, uninitialized. An imple-
 62 mentation may cause *posix_spawn_file_actions_destroy()* to set the object
 63 referenced by *file_actions* to an invalid value. A destroyed spawn file actions

object can be reinitialized using *posix_spawn_file_actions_init()*; the results of otherwise referencing the object after it has been destroyed are undefined.

The *posix_spawn_file_actions_addclose()* function adds a close action to the object referenced by *file_actions* that will cause the file descriptor *fil-des* to be closed [as if *close(fil-des)* had been called] when a new process is spawned using this file actions object.

The *posix_spawn_file_actions_adddup2()* function adds a dup2 action to the object referenced by *file_actions* that will cause the file descriptor *fil-des* to be duplicated as *newfil-des* [as if *dup2(fil-des, newfil-des)* had been called] when a new process is spawned using this file actions object.

The *posix_spawn_file_actions_addopen()* function adds an open action to the object referenced by *file_actions* that will cause the file named by *path* to be opened [as if *open(path, oflag, mode)* had been called, and the returned file descriptor, if not *fil-des*, had been changed to *fil-des*] when a new process is spawned using this file actions object. If *fil-des* was already an open file descriptor, it shall be closed before the new file is opened.

A spawn file actions object, when passed to *posix_spawn()* or *posix_spawnnp()*, shall specify how the set of open file descriptors in the calling process is transformed into a set of potentially open file descriptors for the spawned process. This transformation shall be as if the specified sequence of actions was performed exactly once, in the context of the spawned process (prior to execution of the new process image), in the order in which the actions were added to the object; additionally, when the new process image is executed, any file descriptor (from this new set) which has its FD_CLOEXEC flag set will be closed (see 3.1.6).

Otherwise:

Either the implementation shall support the *posix_spawn_file_actions_init()*, *posix_spawn_file_actions_destroy()*, *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*, and *posix_spawn_file_actions_addopen()* functions as described above, or these functions shall not be provided.

3.1.4.3 Returns

Upon successful completion, the *posix_spawn_file_actions_init()*, *posix_spawn_file_actions_destroy()*, *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*, or *posix_spawn_file_actions_addopen()* operation shall return zero. Otherwise, an error number shall be returned to indicate the error.

3.1.4.4 Errors

For each of the following conditions, if the condition is detected, the *posix_spawn_file_actions_init()*, *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*, or *posix_spawn_file_actions_addopen()* function shall return the corresponding error number:

107 [ENOMEM] Insufficient memory exists to initialize or add to the spawn file
108 actions object.

109 For each of the following conditions, if the condition is detected, the
110 *posix_spawn_file_actions_destroy()*, *posix_spawn_file_actions_addclose()*,
111 *posix_spawn_file_actions_adddup2()*, or *posix_spawn_file_actions_addopen()* func-
112 tion shall return the corresponding error number:

113 [EINVAL] The value specified by *file_actions* is invalid.

114 For each of the following conditions, the *posix_spawn_file_actions_addclose()*,
115 *posix_spawn_file_actions_adddup2()*, or *posix_spawn_file_actions_addopen()* func-
116 tion shall return the corresponding error number:

117 [EBADF] The value specified by *fildes* is negative or greater than or equal
118 to {OPEN_MAX}.

119 It shall not be considered an error for the *fildes* argument passed to the
120 *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*, or
121 *posix_spawn_file_actions_addopen()* functions to specify a file descriptor for which
122 the specified operation could not be performed at the time of the call. Any such
123 error will be detected when the associated file actions object is later used during a
124 *posix_spawn()* or *posix_spawnnp()* operation.

125 **3.1.4.5 Cross-References**

126 *close()*, 6.3.1; *dup2()*, 6.2.1; *open()*, 5.3.1; *posix_spawn()*, 3.1.6; *posix_spawnnp()*,
127 3.1.6;

128 **3.1.5 Spawn Attributes**

129 Functions: *posix_spawnattr_init()*, *posix_spawnattr_destroy()*,
130 *posix_spawnattr_getflags()*, *posix_spawnattr_setflags()*,
131 *posix_spawnattr_getpgroup()*, *posix_spawnattr_setpgroup()*,
132 *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_setschedpolicy()*,
133 *posix_spawnattr_getschedparam()*, *posix_spawnattr_setschedparam()*,
134 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setsigmask()*,
135 *posix_spawnattr_getsigdefault()*, *posix_spawnattr_setsigdefault()*.

136 3.1.5.1 Synopsis

```

137 #include <sys/types.h>
138 #include <signal.h>
139 #include <spawn.h>

140 int posix_spawnattr_init (posix_spawnattr_t *attr);
141 int posix_spawnattr_destroy (posix_spawnattr_t *attr);
142 int posix_spawnattr_getflags (const posix_spawnattr_t *attr,
143                               short *flags);
144 int posix_spawnattr_setflags (posix_spawnattr_t *attr,
145                               short flags);
146 int posix_spawnattr_getpgroup (const posix_spawnattr_t *attr,
147                                pid_t *pgroup);
148 int posix_spawnattr_setpgroup (posix_spawnattr_t *attr,
149                                pid_t pgroup);
150 int posix_spawnattr_getsigmask (const posix_spawnattr_t *attr,
151                                 sigset_t *sigmask);
152 int posix_spawnattr_setsigmask (posix_spawnattr_t *attr,
153                                 const sigset_t *sigmask);
154 int posix_spawnattr_getsigdefault (const posix_spawnattr_t *attr,
155                                   sigset_t *sigdefault);
156 int posix_spawnattr_setsigdefault (posix_spawnattr_t *attr,
157                                   const sigset_t *sigdefault);

158 #include <sched.h>
159 int posix_spawnattr_getschedpolicy (const posix_spawnattr_t *attr,
160                                     int *schedpolicy);
161 int posix_spawnattr_setschedpolicy (posix_spawnattr_t *attr,
162                                     int schedpolicy);
163 int posix_spawnattr_getschedparam (const posix_spawnattr_t *attr,
164                                    struct sched_param *schedparam);
165 int posix_spawnattr_setschedparam (posix_spawnattr_t *attr,
166                                    const struct sched_param *schedparam);

```

167 3.1.5.2 Description

168 If `{_POSIX_SPAWN}` is defined:

169 A spawn attributes object is of type *posix_spawnattr_t* (defined in
170 `<spawn.h>`) and is used to specify the inheritance of process attributes
171 across a spawn operation. This standard does not define comparison or
172 assignment operators for the type *posix_spawnattr_t*.

173 The function *posix_spawnattr_init()* initializes a spawn attributes object
174 *attr* with the default value for all of the individual attributes used by the
175 implementation.

176 Each implementation shall document the individual attributes it uses and
177 their default values unless these values are defined by this standard.

178 The resulting spawn attributes object (possibly modified by setting indivi-
179 dual attribute values) is used to modify the behavior of *posix_spawn()* or
180 *posix_spawnnp()* (see 3.1.6). After a spawn attributes object has been used to
181 spawn a process by a call to a *posix_spawn()* or *posix_spawnnp()*, any func-
182 tion affecting the attributes object (including destruction) does not affect
183 any process that has been spawned in this way.

184 The *posix_spawnattr_destroy()* function destroys a spawn attributes object.
185 The effect of subsequent use of the object is undefined until the object is re-
186 initialized by another call to *posix_spawnattr_init()*. An implementation
187 may cause *posix_spawnattr_destroy()* to set the object referenced by *attr* to
188 an invalid value.

189 The **spawn-flags** attribute is used to indicate which process attributes
190 are to be changed in the new process image when invoking *posix_spawn()*
191 or *posix_spawnnp()*. It is the inclusive OR of zero or more of the flags
192 POSIX_SPAWN_SETPGROUP, POSIX_SPAWN_RESETPIDS,
193 POSIX_SPAWN_SETSIGMASK, and POSIX_SPAWN_SETSIGDEF. In addition,
194 if the Process Scheduling option is supported, the flags
195 POSIX_SPAWN_SETSCHEDULER and POSIX_SPAWN_SETSCHEDPARAM
196 shall also be supported. These flags are defined in **<spawn.h>**. The
197 default value of this attribute shall be with no flags set.

198 The *posix_spawnattr_setflags()* function is used to set the **spawn-flags**
199 attribute in an initialized attributes object referenced by *attr*. The
200 *posix_spawnattr_getflags()* function obtains the value of the **spawn-flags**
201 attribute from the attributes object referenced by *attr*.

202 The **spawn-pgroup** attribute represents the process group to be joined by
203 the new process image in a spawn operation (if POSIX_SPAWN_SETPGROUP
204 is set in the **spawn-flags** attribute). The default value of this attribute
205 shall be zero.

206 The *posix_spawnattr_setpgroup()* function is used to set the **spawn-**
207 **pgroup** attribute in an initialized attributes object referenced by *attr*. The
208 *posix_spawnattr_getpgroup()* function obtains the value of the **spawn-**
209 **pgroup** attribute from the attributes object referenced by *attr*.

210 The **spawn-sigmask** attribute represents the signal mask in effect in the
211 new process image of a spawn operation (if POSIX_SPAWN_SETSIGMASK is
212 set in the **spawn-flags** attribute). The default value of this attribute is
213 unspecified.

214 The *posix_spawnattr_setsigmask()* function is used to set the **spawn-**
215 **sigmask** attribute in an initialized attributes object referenced by *attr*.
216 The *posix_spawnattr_getsigmask()* function obtains the value of the
217 **spawn-sigmask** attribute from the attributes object referenced by *attr*.

218 The **spawn-sigdefault** attribute represents the set of signals to be
219 forced to default signal handling in the new process image (if
220 POSIX_SPAWN_SETSIGDEF is set in the **spawn-flags** attribute). The
221 default value of this attribute shall be an empty signal set.

222 The *posix_spawnattr_setsigdefault()* function is used to set the **spawn-**
 223 **sigdefault** attribute in an initialized attributes object referenced by *attr*.
 224 The *posix_spawnattr_getsigdefault()* function obtains the value of the
 225 **spawn-sigdefault** attribute from the attributes object referenced by
 226 *attr*.

227 Otherwise:

228 Either the implementation shall support the *posix_spawnattr_init()*,
 229 *posix_spawnattr_destroy()*, *posix_spawnattr_getflags()*,
 230 *posix_spawnattr_setflags()*, *posix_spawnattr_getpgroup()*,
 231 *posix_spawnattr_setpgroup()*, *posix_spawnattr_getsigmask()*,
 232 *posix_spawnattr_setsigmask()*, *posix_spawnattr_getsigdefault()*, and
 233 *posix_spawnattr_setsigdefault()* functions as described above, or these func-
 234 tions shall not be provided.

235 If `{_POSIX_SPAWN}` and `{_POSIX_PRIORITY_SCHEDULING}` are both defined:

236 The **spawn-schedpolicy** attribute represents the scheduling policy to be
 237 assigned to the new process image in a spawn operation (if
 238 `POSIX_SPAWN_SETSCHEDULER` is set in the **spawn-flags** attribute).
 239 The default value of this attribute is unspecified.

240 The *posix_spawnattr_setschedpolicy()* function is used to set the **spawn-**
 241 **schedpolicy** attribute in an initialized attributes object referenced by
 242 *attr*. The *posix_spawnattr_getschedpolicy()* function obtains the value of the
 243 **spawn-schedpolicy** attribute from the attributes object referenced by
 244 *attr*.

245 The **spawn-schedparam** attribute represents the scheduling parameters
 246 to be assigned to the new process image in a spawn operation (if
 247 `POSIX_SPAWN_SETSCHEDULER` or `POSIX_SPAWN_SETSCHEDPARAM` is set
 248 in the **spawn-flags** attribute). The default value of this attribute is
 249 unspecified.

250 The *posix_spawnattr_setschedparam()* function is used to set the **spawn-**
 251 **schedparam** attribute in an initialized attributes object referenced by *attr*.
 252 The *posix_spawnattr_getschedparam()* function obtains the value of the
 253 **spawn-schedparam** attribute from the attributes object referenced by
 254 *attr*.

255 Otherwise:

256 Either the implementation shall support the
 257 *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_setschedpolicy()*,
 258 *posix_spawnattr_getschedparam()*, and *posix_spawnattr_setschedparam()*
 259 functions as described above, or these functions shall not be provided.

260 Additional attributes, their default values, and the names of the associated func-
 261 tions to get and set those attribute values are implementation defined.

262 3.1.5.3 Returns

263 If successful, the *posix_spawnattr_init()*, *posix_spawnattr_destroy()*,
 264 *posix_spawnattr_setflags()*, *posix_spawnattr_setpgroup()*,
 265 *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_setschedparam()*,

266 *posix_spawnattr_setsigmask()*, and *posix_spawnattr_setsigdefault()* functions
 267 shall return zero. Otherwise, an error number shall be returned to indicate the
 268 error.

269 If successful, the *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*,
 270 *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_getschedparam()*,
 271 *posix_spawnattr_getsigmask()*, and *posix_spawnattr_getsigdefault()* functions
 272 shall return zero and store the value of the **spawn-flags**, **spawn-pgroup**,
 273 **spawn-schedpolicy**, **spawn-schedparam**, **spawn-sigmask**, or **spawn-**
 274 **sigdefault** attribute of *attr* into the object referenced by the *flags*, *pgroup*,
 275 *schedpolicy*, *schedparam*, *sigmask*, or *sigdefault* parameter, respectively. Other-
 276 wise, an error number shall be returned to indicate the error.

277 **3.1.5.4 Errors**

278 If any of the following conditions occur, the *posix_spawnattr_init()* function shall
 279 return the corresponding error value:

280 [ENOMEM] Insufficient memory exists to initialize the spawn attributes
 281 object.

282 For each of the following conditions, if the condition is detected, the
 283 *posix_spawnattr_destroy()*, *posix_spawnattr_getflags()*,
 284 *posix_spawnattr_setflags()*, *posix_spawnattr_getpgroup()*,
 285 *posix_spawnattr_setpgroup()*, *posix_spawnattr_getschedpolicy()*,
 286 *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_getschedparam()*,
 287 *posix_spawnattr_setschedparam()*, *posix_spawnattr_getsigmask()*,
 288 *posix_spawnattr_setsigmask()*, *posix_spawnattr_getsigdefault()*, and
 289 *posix_spawnattr_setsigdefault()* functions shall return the corresponding error
 290 value:

291 [EINVAL] The value specified by *attr* is invalid.

292 For each of the following conditions, if the condition is detected, the
 293 *posix_spawnattr_setflags()*, *posix_spawnattr_setpgroup()*,
 294 *posix_spawnattr_setschedpolicy()*, *posix_spawnattr_setschedparam()*,
 295 *posix_spawnattr_setsigmask()*, and *posix_spawnattr_setsigdefault()* functions
 296 shall return the corresponding error value:

297 [EINVAL] The value of the attribute being set is not valid.

298 **3.1.5.5 Cross-References**

299 *posix_spawn()*, 3.1.6; *posix_spawnnp()*, 3.1.6.

300 **3.1.6 Spawn a Process**

301 Functions: *posix_spawn()*, *posix_spawnnp()*.

302 3.1.6.1 Synopsis

```

303 #include <sys/types.h>
304 #include <spawn.h>

305 int posix_spawn(pid_t *pid,
306                 const char *path,
307                 const posix_spawn_file_actions_t *file_actions,
308                 const posix_spawnattr_t *attrp,
309                 char * const argv[],
310                 char * const envp[]);

311 int posix_spawnp(pid_t *pid,
312                  const char *file,
313                  const posix_spawn_file_actions_t *file_actions,
314                  const posix_spawnattr_t *attrp,
315                  char * const argv[],
316                  char * const envp[]);

```

317 3.1.6.2 Description

318 If `{_POSIX_SPAWN}` is defined:

319 The *posix_spawn()* and *posix_spawnp()* functions shall create a new process
 320 (child process) from the specified process image. The new process image is
 321 constructed from a regular executable file called the new process image file.

322 When a C program is executed as the result of this call, it shall be entered
 323 as a C language function call as follows:

```

324     int main (int argc, char *argv[]);

```

325 where *argc* is the argument count and *argv* is an array of character
 326 pointers to the arguments themselves. In addition, the variable

```

327     extern char **environ;

```

328 is initialized as a pointer to an array of character pointers to the environ-
 329 ment strings.

330 The argument *argv* is an array of character pointers to null-terminated
 331 strings. The last member of this array shall be a **NULL** pointer and is not
 332 counted in *argc*. These strings constitute the argument list available to the
 333 new process image. The value in *argv[0]* should point to a filename that is
 334 associated with the process image being started by the *posix_spawn()* or
 335 *posix_spawnp()* function.

336 The argument *envp* is an array of character pointers to null-terminated
 337 strings. These strings constitute the environment for the new process
 338 image. The environment array is terminated by a **NULL** pointer.

339 The number of bytes available for the child process's combined argument
 340 and environment lists is `{ARG_MAX}`. The implementation shall specify in
 341 the system documentation (see 1.3.1) whether any list overhead, such as
 342 length words, null terminators, pointers, or alignment bytes, is included in
 343 this total.

344 The *path* argument to *posix_spawn()* is a pathname that identifies the new
 345 process image file to execute.

The *file* parameter to *posix_spawn()* shall be used to construct a pathname that identifies the new process image file. If the *file* parameter contains a slash character, the *file* parameter shall be used as the pathname for the new process image file. Otherwise, the path prefix for this file shall be obtained by a search of the directories passed as the environment variable **PATH** (see 2.6). If this environment variable is not defined, the results of the search are implementation defined.

If *file_actions* is a **NULL** pointer, then file descriptors open in the calling process shall remain open in the child process, except for those whose close-on-exec flag **FD_CLOEXEC** is set (see 6.5.2 and 6.5.1). For those file descriptors that remain open, all attributes of the corresponding open file descriptions, including file locks (see 6.5.2), shall remain unchanged.

If *file_actions* is not **NULL**, then the file descriptors open in the child process shall be those open in the calling process as modified by the spawn file actions object pointed to by *file_actions* and the **FD_CLOEXEC** flag of each remaining open file descriptor after the spawn file actions have been processed. The effective order of processing the spawn file actions shall be

1. The set of open file descriptors for the child process shall initially be the same set as is open for the calling process. All attributes of the corresponding open file descriptions, including file locks (see 6.5.2), shall remain unchanged.
2. The signal mask and the effective user and group IDs for the child process shall be changed as specified in the attributes object referenced by *attrp*.
3. The file actions specified by the spawn file actions object shall be performed in the order in which they were added to the spawn file actions object.
4. Any file descriptor that has its **FD_CLOEXEC** flag set (see 6.5.2) shall be closed.

The *posix_spawnattr_t* spawn attributes object type is defined in **<spawn.h>**. It shall contain at least the attributes described in 3.1.5.

If the **POSIX_SPAWN_SETPGROUP** flag is set in the **spawn-flags** attribute of the object referenced by *attrp* and the **spawn-pgroup** attribute of the same object is non-zero, then the child's process group shall be as specified in the **spawn-pgroup** attribute of the object referenced by *attrp*.

As a special case, if the **POSIX_SPAWN_SETPGROUP** flag is set in the **spawn-flags** attribute of the object referenced by *attrp* and the **spawn-pgroup** attribute of the same object is set to zero, then the child shall be in a new process group with a process group ID equal to its process ID.

If the **POSIX_SPAWN_SETPGROUP** flag is not set in the **spawn-flags** attribute of the object referenced by *attrp*, the new child process shall inherit the parent's process group.

If **{_POSIX_PRIORITY_SCHEDULING}** is defined and the **POSIX_SPAWN_SETSCHEDPARAM** flag is set in the **spawn-flags** attribute of the object referenced by *attrp*, but **POSIX_SPAWN_SETSCHEDULER** is not set, the new process image shall initially have the scheduling policy of the calling process with the scheduling parameters specified in the **spawn-**

393 **schedparam** attribute of the object referenced by *attrp*.

394 If `{_POSIX_PRIORITY_SCHEDULING}` is defined and the
395 `POSIX_SPAWN_SETSCHEDULER` flag is set in **spawn-flags** attribute of
396 the object referenced by *attrp* (regardless of the setting of the
397 `POSIX_SPAWN_SETSCHEDPARAM` flag), the new process image shall ini-
398 tially have the scheduling policy specified in the **spawn-schedpolicy**
399 attribute of the object referenced by *attrp* and the scheduling parameters
400 specified in the **spawn-schedparam** attribute of the same object.

401 The `POSIX_SPAWN_RESETIDS` flag in the **spawn-flags** attribute of the
402 object referenced by *attrp* governs the effective user ID of the child process.
403 If this flag is not set, the child process inherits the parent process's effective
404 user ID. If this flag is set, the child process's effective user ID is reset to the
405 parent's real user ID. In either case, if the set-user-ID mode bit of the new
406 process image file is set, the effective user ID of the child process will
407 become that file's owner ID before the new process image begins execution.

408 The `POSIX_SPAWN_RESETIDS` flag in the **spawn-flags** attribute of the
409 object referenced by *attrp* also governs the effective group ID of the child
410 process. If this flag is not set, the child process inherits the parent process's
411 effective group ID. If this flag is set, the child process's effective group ID is
412 reset to the parent's real group ID. In either case, if the set-group-ID mode
413 bit of the new process image file is set, the effective group ID of the child
414 process will become that file's group ID before the new process image begins
415 execution.

416 If the `POSIX_SPAWN_SETSIGMASK` flag is set in the **spawn-flags** attri-
417 bute of the object referenced by *attrp*, the child process shall initially have
418 the signal mask specified in the **spawn-sigmask** attribute of the object
419 referenced by *attrp*.

420 If the `POSIX_SPAWN_SETSIGDEF` flag is set in the **spawn-flags** attribute
421 of the object referenced by *attrp*, the signals specified in the **spawn-**
422 **sigdefault** attribute of the same object shall be set to their default
423 actions in the child process. Signals set to their default actions in the
424 parent process shall be set to their default actions in the child process.

425 Signals set to be caught by the calling process shall be set to their default
426 actions in the child process.

427 Signals set to be ignored by the calling process image shall be set to be
428 ignored by the child process, unless otherwise specified by the
429 `POSIX_SPAWN_SETSIGDEF` flag being set in the **spawn-flags** attribute of
430 the object referenced by *attrp* and the signals being indicated in the
431 **spawn-sigdefault** attribute of the object referenced by *attrp*.

432 If the value of the *attrp* pointer is **NULL**, then the default values are used.

433 All process attributes other than those influenced by the attributes set in
434 the object referenced by *attrp* as specified above or by the file descriptor
435 manipulations specified in *file_actions* shall appear in the new process
436 image as though *fork()* had been called to create a child process and then a
437 member of the *exec* family of functions had been called by the child process
438 to execute the new process image.

439 If the Threads option is supported, then it is implementation defined
 440 whether the fork handlers are run when *posix_spawn()* or *posix_spawnp()*
 441 is called.

442 Otherwise :

443 Either the implementation shall support the *posix_spawn()* and
 444 *posix_spawnp()* functions as described above, or these functions shall not be
 445 provided.

446 3.1.6.3 Returns

447 Upon successful completion, the *posix_spawn()* or *posix_spawnp()* operation shall
 448 return the process ID of the child process to the parent process, in the variable
 449 pointed to by a non-**NULL** *pid* argument, and shall return zero as the function
 450 return value. Otherwise, no child process shall be created, the value stored into
 451 the variable pointed to by a non-**NULL** *pid* is unspecified, and the corresponding
 452 error value shall be returned as the function return value. If the *pid* argument is
 453 the **NULL** pointer, the process ID of the child is not returned to the caller.

454 3.1.6.4 Errors

455 For each of the following conditions, if the condition is detected, the *posix_spawn()*
 456 or *posix_spawnp()* function shall fail and post the corresponding status value or, if
 457 the error occurs after the calling process successfully returns from the
 458 *posix_spawn()* or *posix_spawnp()* function, shall cause the child process to exit
 459 with exit status 127:

460 [EINVAL] The value specified by *file_actions* or *attrp* is invalid.

461 If *posix_spawn()* or *posix_spawnp()* fails for any of the reasons that would cause
 462 *fork()* or one of the *exec* family of functions to fail, an error value shall be returned
 463 as described by *fork()* and *exec*, respectively (or, if the error occurs after the cal-
 464 ling process successfully returns, the child process exits with exit status 127).

465 If POSIX_SPAWN_SETPGROUP is set in the **spawn-flags** attribute of the object
 466 referenced by *attrp* and *posix_spawn()* or *posix_spawnp()* fails while changing the
 467 child's process group, an error value shall be returned as described by *setpgid()*
 468 (or, if the error occurs after the calling process successfully returns, the child pro-
 469 cess exits with exit status 127).

470 If `{_POSIX_PRIORITY_SCHEDULING}` is defined, if
 471 POSIX_SPAWN_SETSCHEDPARAM is set and POSIX_SPAWN_SETSCHEDULER is
 472 not set in the **spawn-flags** attribute of the object referenced by *attrp*, and if
 473 *posix_spawn()* or *posix_spawnp()* fails for any of the reasons that would cause
 474 *sched_setparam()* to fail, an error value shall be returned as described by
 475 *sched_setparam()* (or, if the error occurs after the calling process successfully
 476 returns, the child process exits with exit status 127).

477 If `{_POSIX_PRIORITY_SCHEDULING}` is defined, if
 478 POSIX_SPAWN_SETSCHEDULER is set in the **spawn-flags** attribute of the
 479 object referenced by *attrp*, and if *posix_spawn()* or *posix_spawnp()* fails for any of
 480 the reasons that would cause *sched_setscheduler()* to fail, an error value shall be
 481 returned as described by *sched_setscheduler()* (or, if the error occurs after the cal-
 482 ling process successfully returns, the child process exits with exit status 127).

483 If the *file_actions* argument is not **NULL** and specifies any close, dup2, or open
 484 actions to be performed and if *posix_spawn()* or *posix_spawnnp()* fails for any of the
 485 reasons that would cause *close()*, *dup2()*, or *open()* to fail, an error value shall be
 486 returned as described by *close()*, *dup2()*, and *open()*, respectively (or, if the error
 487 occurs after the calling process successfully returns, the child process exits with
 488 exit status 127). An open file action may, by itself, result in any of the errors
 489 described by *close()* or *dup2()*, in addition to those described by *open()*.

490 3.1.6.5 Cross-References

491 *alarm()*, 3.4.1; *chmod()*, 5.6.4; *close()*, 6.3.1; *dup2()*, 6.2.1; *exec*, 3.1.2; *_exit()*, 3.2.2;
 492 *fcntl()*, 6.5.2; *fork()*, 3.1.1; *kill()*, 3.3.2; *open()*, 5.3.1;
 493 *posix_spawn_file_actions_init()*, 3.1.4; *posix_spawn_file_actions_destroy()*, 3.1.4;
 494 *posix_spawn_file_actions_addclose()*, 3.1.4; *posix_spawn_file_actions_adddup2()*,
 495 3.1.4; *posix_spawn_file_actions_addopen()*, 3.1.4; *posix_spawnattr_init()*, 3.1.5;
 496 *posix_spawnattr_destroy()*, 3.1.5; *posix_spawnattr_getflags()*, 3.1.5;
 497 *posix_spawnattr_setflags()*, 3.1.5; *posix_spawnattr_getpgroup()*, 3.1.5;
 498 *posix_spawnattr_setpgroup()*, 3.1.5; *posix_spawnattr_getschedpolicy()*, 3.1.5;
 499 *posix_spawnattr_setschedpolicy()*, 3.1.5; *posix_spawnattr_getschedparam()*, 3.1.5;
 500 *posix_spawnattr_setschedparam()*, 3.1.5; *posix_spawnattr_getsigmask()*, 3.1.5;
 501 *posix_spawnattr_setsigmask()*, 3.1.5; *posix_spawnattr_getsigdefault()*, 3.1.5;
 502 *posix_spawnattr_setsigdefault()*, 3.1.5; *sched_setparam()*, 13.3.1;
 503 *sched_setscheduler()*, 13.3.3; *setpgid()*, 4.3.3; *setuid()*, 4.2.2; *stat()*, 5.6.2; *times()*,
 504 4.5.2; *wait*, 3.2.1.

505 3.2 Process Termination

506 3.2.1 Wait for Process Termination

507 3.2.1.2 Wait for Process Termination — Description

508 ⇒ **3.2.1.2 Wait for Process Termination — Description** Add the following
 509 paragraphs after the definition of the *WSTOPSIG(stat_val)* macro:

510 It is unspecified whether the status value returned by calls to *wait()* or *wait-*
 511 *pid()* for processes created by *posix_spawn()* or *posix_spawnnp()* may indicate a
 512 *WIFSTOPPED(stat_val)* before subsequent calls to *wait()* or *waitpid()* indicate
 513 *WIFEXITED(stat_val)* as the result of an error detected before the new process
 514 image starts executing.

515 It is unspecified whether the status value returned by calls to *wait()* or *wait-*
 516 *pid()* for processes created by *posix_spawn()* or *posix_spawnnp()* may indicate a
 517 *WIFSIGNALED(stat_val)* if a signal is sent to the parent's process group after
 518 *posix_spawn()* or *posix_spawnnp()* is called.

