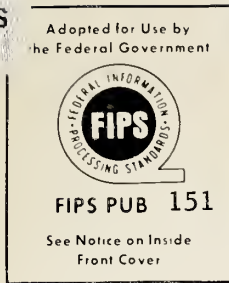


**NATIONAL INSTITUTE OF STANDARDS &
TECHNOLOGY**
Research Information Center
Gaithersburg, MD 20899





CA11102 966587



Portable Operating System Interface for Computer Environments

Sponsor

Technical Committee on Operating Systems
of the
IEEE Computer Society

Unapproved Draft. All rights reserved by IEEE. Do not specify or claim conformance to this document.

©1988 by

**The Institute of Electrical and Electronics Engineers, Inc
345 East 47th Street, New York, NY 10017, USA**

*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.*

JK
468
A8A3
#151 p.2
1988

157C

JK 465. ADAS NO. 157 P. 1500 112

P1003.1 Draft

Portable Operating System Interface for Computer Environments

Sponsor
Technical Committee on Operating Systems
of the
IEEE
Computer Society

P1003.1 / DRAFT 12
October 12, 1987

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

Research Information Center
National Institute of Standards
and Technology
Gaithersburg, Maryland 20899

This standard has been adopted for Federal Government use.

Details concerning its use within the Federal Government are contained in Federal Information Processing Standards Publication 151, POSIX: Portable Operating System Interface for Computer Environments. For a complete list of publications available in the Federal Information Processing Standards Series, write to the Standards Processing Coordinator (ADP), National Computer and Telecommunications Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899.

USER NOTE: Draft 12 of IEEE 1003.1 is not the most current version of this standard and is not identical to IEEE Std 1003.1-1988. IEEE Std 1003.1-1988 is the published version of Draft 13, which was approved by the IEEE Standards Board on August 22, 1988.

Foreword

1

2 (This Foreword is not a part of IEEE Std 1003.1, IEEE Standard Portable Operating c
3 System Interface for Computer Environments.) c

4 The purpose of this standard is to define a standard operating system interface and 8
5 environment based on the UNIX* Operating System documentation to support 8
6 application portability at the source level. This is intended for systems implementors and
7 applications software implementors.

8 In its present form, the standard focuses primarily on the C Language interface to the
9 operating system.

10 IEEE Std 1003.1 is the first of a group of proposed standards known colloquially, and c
11 collectively, as POSIX†. The other POSIX standards are described in Appendix A. c

12 Organization of the Standard 8

13 The standard is divided into four parts: 8

14 • Statement of scope (Chapter 1) 8

15 • Definitions and global concepts (Chapter 2) 8

16 • The various interface facilities (Chapters 3 through 9) 8

17 • Data interchange format (Chapter 10) 8

18 This foreword and the appendices are not considered part of the standard.

19 Most of the sections describe a single service interface. The C Language binding for the c
20 service interface is given in the subsection labeled **Synopsis**. The **Description** c
21 subsection provides a specification of the operation performed by the service interface. c
22 Some examples may be provided to illustrate the interfaces described. In most cases
23 there are also **Returns** and **Errors** subsections specifying return values and possible
24 error conditions. References are used to direct the reader to other related sections. 9
25 Additional material to complement sections in the standard may be found in **Rationale** 9
26 **and Notes**, Appendix B. This appendix provides historical perspectives into the 9
27 technical choices made by the 1003.1 Working Group. It also provides information to 9
28 emphasize consequences of the interfaces described in the corresponding section of the 9
29 standard. 9

* UNIX is a registered trademark of AT&T.

† POSIX is pronounced *pahz-icks*, similar to *positive*.

30 In publishing this standard, both the IEEE and the 1003.1 Working Group simply intend
31 to provide a yardstick against which various operating system implementations can be
32 measured for conformance. It is *not* the intent of either the IEEE or the 1003.1 Working
33 Group to measure or rate any products, to reward or sanction any vendors of products for
34 conformance or lack of conformance to this standard, or to attempt to enforce this
35 standard by these or any other means. The responsibility for determining the degree of
36 conformance or lack thereof with this standard rests solely with the individual who is
37 evaluating the product claiming to be in conformance with the standard. (See 9
38 Verification Testing §A.2.3 for additional information on this subject.) 9

39 Base Documents

40 The various interface facilities described herein are based on the *1984 /usr/group*
41 *Standard* derived and published by the /usr/group Standards Committee, Santa Clara,
42 California. The *1984 /usr/group Standard*, and subsequent work of the 1003.1 Working B
43 Group is largely based on UNIX Seventh Edition, System III, System V, 4.2BSD, and
44 4.3BSD documentation, but wherever possible, compatibility with other UNIX-derived
45 systems and compatible systems has been maintained.

46 The IEEE is grateful to both AT&T and /usr/group for permission to use their materials.
47 9

48 Extensions and Supplements to this Standard 9

49 Activities to extend this standard to address additional requirements are in progress and 9
50 similar efforts can be anticipated in the future. This is an outline of how these extensions 9
51 will be incorporated, and also how users of this document can keep track of that status. 9

52 Extensions are approved as "Supplements" to this document, following the IEEE 9
53 Standards Procedures. 9

54 Approved Supplements are published separately and distributed with orders from the 9
55 IEEE for this document until the full document is reprinted and such supplements are 9
56 incorporated in their proper positions. 9

57 If you have any question about the completeness of your version, you may contact the 9
58 IEEE Computer Society (*phone # to be provided*) or the IEEE Standards Office (*phone #* 9
59 *to be provided*) to determine what supplements have been published. Published 9
60 supplements will be available for a modest fee. 9

61 Supplements are numbered in the same format as the main document, and with unique 9
62 positions as either subsections or main sections. A supplement may include new 9
63 subsections in various sections of the main document as well as new main sections. 9
64 Supplements may include new sections in already approved supplements. However, the 9
65 overall numbering shall be unique so that two supplements do not use the same numbers 9
66 unless one replaces the other. 9

67 Supplements may contain either required functions or optional facilities. Supplements 9
68 may add additional conformance requirements (see Conformance §2.2) defining new 9

classes of conforming systems or applications. 9

70 It is desirable, but perhaps not avoidable, that supplements do not change the 9
71 functionality of the already defined facilities. 9

72 Supplements are not used to provide a general update of the standard. This is done 9
73 through the review procedure as specified by the IEEE. 9

74 The following areas are under active consideration at this time, or are expected to 9
75 become active in the near future. 9

- 76 • Shell and Utility facilities — P1003.2 (see Shell and Utilities §A.2.2); 9
- 77 • Verification Testing — P1003.3 (see Verification Testing §A.2.3); 9
- 78 • Real Time facilities — P1003.4 (see Real Time Extensions §A.2.4); c
- 79 • Secure/Trusted System considerations; c
- 80 • FORTRAN Language bindings; c
- 81 • Ada* Language bindings; c
- 82 • Language-independent service descriptions; c
- 83 • An overall guide to POSIX-based or related Open Systems standards. c

84 (See Appendix A for additional information.) If you have interest in participating in the 9
85 working groups addressing these issues, please send your name, address, and phone 9
86 number to the: 9

87 Secretary, IEEE Standards Board 9
88 Institute of Electrical and Electronics Engineers, Inc. 9
89 345 East 47th Street 9
90 New York, NY 10017 9

91 and ask to have this forwarded to the chairperson of the appropriate P1003 working 9
92 group. 9

* Ada is a trademark of the Department of Defense.

93 *Editor's Notes* c

94 *This section will not appear in the final document. It is used for editorial comments* c
95 *concerning Draft 12.* c

96 *This draft uses small numbers in the right margin in lieu of change bars. "8" denotes* c
97 *changes from Draft 7 (the Trial Use Standard) to Draft 8. "9" denotes changes from* c
98 *Draft 8 to Draft 9. "A" denotes changes from Draft 9 to Draft 10 (in hex). "B" denotes* c
99 *changes from Draft 10 to Draft 11 (in hex). "C" denotes changes from Draft 11 to Draft* c
100 *12 (in hex). Deleted text uses the same symbols, but will generally be noted by a blank* c
101 *line containing only the change symbol. It should be noted that, due to the algorithms* c
102 *used by troff, some change symbols are overlaid by a following change on the same line,* c
103 *and are therefore obscured. For the future, we will continue hexadecimally and hope* c
104 *that Full Use is achieved before Draft 16. The Full Use standard will have neither* c
105 *change marks or line numbers. The correctness or format of these symbols are not* c
106 *ballotable issues.* c

107 *All of the header paragraphs in the Errors sections have changed slightly ("shall return* c
108 *-1" replaces "shall fail"); these changes are not marked.* c

109 *Please report typographical errors and editorial changes directly to:* c

110 Hal Jespersen c
111 UniSoft Corporation c
112 6121 Hollis Street c
113 Emeryville, CA 94608-2092 c
114 (415) 420-6448 c
115 UUCP: {uunet,amdahl,sun}!unisoft!hlj c

116 *(Electronic mail is preferred.)* c

117 IEEE Std 1003.1 was prepared by the 1003.1 Working Group, sponsored by the Technical
118 Committee on Operating Systems of the IEEE Computer Society.

119 At the time this standard was approved, the membership of the 1003.1 Working Group
120 was as follows:

121 *Editor's Note: This list will be included in the final printed standard.*

Steering Committee			
122			B
123			B
124	Joseph Boykin	<i>Chair, TCOS</i>	C
125	James Isaak	<i>Chair, P1003.1</i>	C
126	Hal Jespersen	<i>Technical Editor, P1003.1</i>	C
127	Shane P. McCarron	<i>Secretary, P1003.1</i>	C
128	Donn S. Terry	<i>Co-Chair, P1003.1</i>	C

Working Group			
129			
130	<i>Name</i>	<i>Name</i>	<i>Name</i>
131	<i>Name</i>	<i>Name</i>	<i>Name</i>
132	<i>Name</i>	<i>Name</i>	<i>Name</i>

133 The following persons were members of the 1003.1 Balloting Group that approved the
134 standard for submission to the IEEE Standards Board:

135	Heinz Lycklama	<i>/usr/group Institutional Representative</i>	C
136	Michael Lambert	<i>X/OPEN Institutional Representative</i>	C
137	John S. Quarterman	<i>USENIX Institutional Representative</i>	C

138 *Editor's Note: This list will be included in the final printed standard.*

139	<i>Name</i>	<i>Name</i>	<i>Name</i>
140	<i>Name</i>	<i>Name</i>	<i>Name</i>
141	<i>Name</i>	<i>Name</i>	<i>Name</i>

Contents

SECTION	PAGE
1. Scope	17
2. Definitions and General Requirements	19
2.1 Terminology	19
2.2 Conformance	20
2.3 General Terms	22
2.4 General Concepts	30
2.5 Error Numbers	32
2.6 Primitive System Data Types	37
2.7 Environment Description	37
2.8 C Language Definitions	39
2.9 Numerical Limits	39
2.10 Symbolic Constants	43
3. Process Primitives	47
3.1 Process Creation	47
3.1.1 Process Creation	47
3.1.2 Execute a File	49
3.2 Process Termination	52
3.2.1 Wait for Process Termination	53
3.2.2 Terminate a Process	55
3.3 Signals	57
3.3.1 Signal Names	57
3.3.2 Send a Signal to a Process	62
3.3.3 Manipulate Signal Sets	64
3.3.4 Examine and Change Signal Action	65
3.3.5 Examine and Change Blocked Signals	67
3.3.6 Examine Pending Signals	68
3.3.7 Wait for a Signal	69
3.4 Timer Operations	70
3.4.1 Process Alarm Clock	70
3.4.2 Suspend Process Execution	71
3.4.3 Delay Process Execution	72
4. Process Environment	73
4.1 Process Identification	73
4.1.1 Get Process and Parent Process IDs	73
4.2 User Identification	73
4.2.1 Get Real User, Effective User, Real Group, and Effective Group IDs	73

SECTION	PAGE
4.2.2 Set User and Group IDs	74
4.2.3 Get Supplementary Group IDs	75
4.2.4 Get User Name	76
4.3 Process Groups	78
4.3.1 Get Process Group ID	78
4.3.2 Set Process Group ID	78
4.3.3 Set Process Group ID for Job Control	79
4.4 System Identification	80
4.4.1 System Name	80
4.5 Time	81
4.5.1 Get System Time	81
4.5.2 Process Times	82
4.6 Environment Variables	83
4.6.1 Environment Access	83
4.7 Terminal Identification	84
4.7.1 Generate Terminal Pathname	84
4.7.2 Determine Terminal Device Name	85
4.8 Configurable System Variables	85
4.8.1 Get Configurable System Variables	85
5. Files and Directories	87
5.1 Directories	87
5.1.1 Format of Directory Entries	87
5.1.2 Directory Operations	88
5.2 Working Directory	90
5.2.1 Change Current Working Directory	90
5.2.2 Working Directory Pathname	91
5.3 General File Creation	92
5.3.1 Open a File	92
5.3.2 Create a New File or Rewrite an Existing One	95
5.3.3 Set File Creation Mask	95
5.3.4 Link to a File	96
5.4 Special File Creation	97
5.4.1 Make a Directory	97
5.4.2 Make a FIFO Special File	99
5.5 File Removal	100
5.5.1 Remove Directory Entries	100
5.5.2 Remove a Directory	102
5.5.3 Rename a File	103
5.6 File Characteristics	106
5.6.1 File Characteristics: Header File and Data Structure	106
5.6.2 Get File Status	108
5.6.3 File Accessibility	110
5.6.4 Change File Modes	111

SECTION	PAGE
5.6.5	Change Owner and Group of a File 112
5.6.6	Set File Access and Modification Times 114
5.7	Configurable Pathname Variables 116
5.7.1	Get Configurable Pathname Variables 116
6.	Input and Output Primitives 119
6.1	Pipes 119
6.1.1	Create an Inter-Process Channel 119
6.2	File Descriptor Manipulation 120
6.2.1	Duplicate an Open File Descriptor 120
6.3	File Descriptor Deassignment 121
6.3.1	Close a File 121
6.4	Input and Output 122
6.4.1	Read from a File 122
6.4.2	Write to a File 124
6.5	Control Operations on Files 127
6.5.1	Data Definitions for File Control Operations 127
6.5.2	File Control 128
6.5.3	Reposition Read/Write File Offset 133
7.	Device- and Class-Specific Functions 135
7.1	General Terminal Interface 135
7.1.1	Interface Characteristics 135
7.1.1.1	Description 135
7.1.1.2	Opening a Terminal Device File 135
7.1.1.3	Process Groups 135
7.1.1.4	The Controlling Terminal 136
7.1.1.5	Job Access Control 136
7.1.1.6	Input Processing and Reading Characters 137
7.1.1.7	Canonical Mode Input Processing 138
7.1.1.8	Non-Canonical Mode Input Processing 138
7.1.1.9	Writing Characters and Output Processing 140
7.1.1.10	Special Characters 140
7.1.1.11	Modem Disconnect 141
7.1.1.12	Closing a Terminal Device File 141
7.1.2	Settable Parameters 142
7.1.2.1	Synopsis 142
7.1.2.2	<i>termios</i> Structure 142
7.1.2.3	Input Modes 142
7.1.2.4	Output Modes 144
7.1.2.5	Control Modes 144
7.1.2.6	Local Modes 147
7.1.2.7	Special Control Characters 148

SECTION	PAGE
7.2 General Terminal Interface Control Functions	149
7.2.1 Get and Set State	149
7.2.2 Line Control Functions	151
7.2.3 Get Distinguished Process Group ID	153
7.2.4 Set Distinguished Process Group ID	154
8. C Language Library	155
8.1 Referenced C Language Routines	155
8.1.1 Extensions to <i>asctime()</i> Function	156
8.1.2 Extensions to <i>setlocale()</i> Function	158
8.2 FILE-Type C Language Functions	160
8.2.1 Map a Stream Pointer to a File Descriptor	160
8.2.2 Open a Stream on a File Descriptor	161
8.3 Other C Language Functions	162
8.3.1 Non-Local Jumps	162
8.3.2 Specify Signal Handling	163
9. System Databases	165
9.1 System Databases	165
9.2 Database Access	166
9.2.1 Group Database Access	166
9.2.2 User Database Access	167
10. Data Interchange Format	169
10.1 Archive/Interchange File Format	169
10.1.1 <i>cpio</i> Archive Format	169
10.1.2 Multiple Volumes	173
APPENDICES	
A. Related Standards	175
A.1 Related Standards — Open System Architecture	175
A.2 Standards Closely Related to the 1003.1 Document	176
A.2.1 C Language Standard	176
A.2.2 Shell and Utilities	176
A.2.3 Verification Testing	178
A.2.4 Real Time Extensions	178
A.2.5 Language Standards	178
A.2.6 Networking Standards	178
A.2.7 Graphics Standards	179
A.2.8 Data Base Standards	179
A.3 Industry Open Systems Publications	180
A.4 US Government Standards	180
A.4.1 Federal Information Processing Standards (FIPS)	180
A.4.2 Trusted Systems	180

SECTION	PAGE
B. Rationale and Notes	181
B.1 Introduction	182
B.1.1 Scope	183
B.1.2 Purpose	183
B.1.2.1 Application Oriented	183
B.1.2.2 Interface, Not Implementation	183
B.1.2.3 Source, Not Object, Portability	184
B.1.2.4 The C Language and X3J11	184
B.1.2.5 No Super-User, No System Administration	184
B.1.2.6 Minimal Interface, Minimally Defined	184
B.1.2.7 Broadly Implementable	184
B.1.2.8 Minimal Changes to Historical Implementations	185
B.1.2.9 Minimal Changes to Existing Application Code	185
B.1.2.10 IEEE Consensus Process	185
B.1.2.11 IEEE Balloting Process	186
B.1.3 Base Documents	188
B.1.3.1 Related Standards and Documents	188
B.1.3.2 Historical Implementations	188
B.1.3.3 Specific Derivations	189
B.1.3.4 Working Documents	190
B.1.4 C Language, X3J11, and P1003.1	191
B.1.4.1 Solely by P1003.1.	192
B.1.4.2 Solely by X3J11.	192
B.1.4.3 By Neither P1003.1 nor X3J11.	193
B.1.4.4 Base by P1003.1, Additions by X3J11.	193
B.1.4.5 Base by X3J11, Additions by P1003.1.	193
B.1.4.6 Related Functions by Both.	193
B.1.5 Organization	194
B.1.5.1 Organization of the Standard	194
B.1.5.2 Organization of this Appendix	195
B.1.5.3 Typographical Conventions	195
B.2 Definitions and General Requirements	196
B.2.1 Terminology	196
B.2.2 Conformance	197
B.2.3 General Terms	200
B.2.4 General Concepts	204
B.2.5 Error Numbers	207
B.2.6 Primitive System Data Types	209
B.2.7 Environment Description	210
B.2.8 C Language Definitions	211

SECTION	PAGE
B.2.9 Numerical Limits	211
B.2.10 Symbolic Constants	215
B.3 Process Primitives	216
B.3.1 Process Creation	216
B.3.2 Process Termination	217
B.3.3 Signals	220
B.3.4 Timer Operations	226
B.4 Process Environment	227
B.4.1 Process Identification	227
B.4.2 User Identification	227
B.4.3 Process Groups	229
B.4.4 System Identification	230
B.4.5 Time	230
B.4.6 Environment Variables	231
B.4.7 Terminal Identification	231
B.4.8 Configurable System Variables	232
B.5 Files and Directories	233
B.5.1 Directories	233
B.5.2 Working Directory	235
B.5.3 General File Creation	236
B.5.4 Special File Creation	237
B.5.5 File Removal	237
B.5.6 File Characteristics	238
B.5.7 Configurable Pathname Variables	240
B.6 Input and Output Primitives	241
B.6.1 Pipes	242
B.6.2 File Descriptor Manipulation	243
B.6.3 File Descriptor Deassignment	243
B.6.4 Input and Output	243
B.6.5 Control Operations on Files	247
B.7 Device- and Class-Specific Functions	250
B.7.1 General Terminal Interface	252
B.7.2 General Terminal Interface Control Functions	255
B.8 C Language Library	256
B.8.1 Referenced C Language Routines	256
B.8.2 FILE-Type C Language Functions	260
B.8.3 Other C Language Functions	260
B.9 System Databases	261
B.9.1 System Databases	261
B.9.2 Database Access	262
B.10 Data Interchange Format	262
B.10.1 Archive/Interchange File Format	262
B.11 Bibliographic Notes	268

SECTION	PAGE
B.11.1 Related Standards	268
B.11.2 Historical Implementations	269
B.11.3 Historical Application Programming Tutorials	271
C. Comparison to <i>System V Interface Definition</i>	273
C.1 Overall Contents	274
C.1.1 Operating System Primitives	274
C.1.2 Library Routines	274
C.1.3 Special Files	275
C.1.4 Minimal Directory Tree Structure	275
C.1.5 Multiple Groups	275
C.1.6 Job Control	275
C.1.7 Enhanced Signals	275
C.1.8 Configurable System Variables	276
C.1.9 Terminal I/O	276
C.2 Specific Differences	276
C.2.1 Error Numbers	276
C.2.2 General Terms	277
C.2.3 Data Types	277
C.2.4 Environment Variables	277
C.2.5 fork()	277
C.2.6 exec()	278
C.2.7 wait()	278
C.2.8 _exit()	278
C.2.9 <signal.h>	279
C.2.10 kill()	279
C.2.11 signal()	279
C.2.12 times()	280
C.2.13 open()	280
C.2.14 unlink()	280
C.2.15 rmdir()	280
C.2.16 <sys/stat.h>	281
C.2.17 access()	281
C.2.18 chown()	281
C.2.19 utime()	281
C.2.20 close()	281
C.2.21 read()	282
C.2.22 write()	282
C.2.23 <fcntl.h>	283
C.2.24 fcntl()	283
C.2.25 lseek()	283
C.2.26 Terminal I/O	283
D. Alternative Archive/Data Interchange Format	285

SECTION	PAGE
D.1 Extended <code>tar</code> Format	285
D.1.1 References	289
E. Alternative <code>wait()</code> Functions	291
E.1 Process Termination	291
E.1.1 Wait for Process Termination	291
Identifier Index	295
Topical Index	301

Portable Operating System Interface for Computer Environments

c

1. Scope

1 This standard defines a standard operating system interface and environment to support 8
2 application portability at the source code level. It is intended to be used by both
3 application developers and system implementors.

4 Initially, the focus of the standard will be on the C language interface. In future c
5 revisions, this will be divided into several parts. The first part will provide a functional c
6 definition of the service interfaces. The following parts will specify the binding between c
7 these service interfaces and specific programming languages, with the second part c
8 describing the C language binding. c

9 This effort entails four major components: c

- 10 1. Definitions for terminology and objects referred to in the standard (in the 8
11 case of objects, their structure, operations that modify objects, and the
12 effects of these operations);
- 13 2. System service interfaces and subroutines; c
- 14 3. C language binding for the system services; c
- 15 4. Interface issues, including portability, error handling, and recovery. 9

16 The following areas are outside of the scope of this standard: 8

- 17 • User interface (shell) and associated commands
- 18 • Network protocols
- 19 • Graphics interfaces
- 20 • Data base management system interfaces
- 21 • Record I/O considerations

- 22 • Object or binary code portability
- 23 (See Appendix A for information about ongoing efforts in some of these areas.)
- 24 This standard describes the external characteristics and facilities that are of importance to
25 applications developers, rather than on the internal construction techniques employed to
26 achieve these capabilities. Special emphasis is placed on those functions and facilities
27 that are needed in a wide variety of commercial applications.
- 28 This standard has been defined exclusively at the source code level. The objective is that
29 a Strictly Conforming Application source program can be compiled to execute on a
30 conforming implementation.

A
C
C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

2. Definitions and General Requirements

B

1 2.1 Terminology

2 The following terms are used in this standard:

A

3 implementation defined

A

4 A value or behavior is **implementation defined** if the implementation defines and
5 documents the requirements for correct program construct and correct data.

A

6 may

A

7 With respect to implementations, the word **may** is to be interpreted as an optional
8 feature that is not required in this standard and can be provided. With respect to
9 **Strictly Conforming Applications**, the word **may** means that the optional feature
10 shall not be used.

A

A

11 shall

A

12 In this standard, the word **shall** is to be interpreted as a requirement on the
13 implementation or on **Strictly Conforming Applications**, where appropriate.

A

A

14 should

A

15 With respect to implementations, the word **should** is to be interpreted as an
16 implementation recommendation, but not a requirement. With respect to
17 applications, the word **should** is to be interpreted as recommended programming
18 practice for applications and a requirement for **Strictly Conforming Applications**.

A

A

A

A

A

19 undefined

A

20 A value or behavior is **undefined** if the standard imposes no portability
21 requirements for erroneous program construct, erroneous data, or use of an
22 indeterminate value.

A

A

23 unspecified

A

24 A value or behavior is **unspecified** if the standard imposes no portability
25 requirements for a correct program construct or correct data.

A

26 2.2 Conformance

27 2.2.1 Implementation Conformance

28 2.2.1.1 Requirements

29 A conforming implementation shall meet all of the following criteria:

30 The system shall support all required interfaces defined within this standard. c
31 These interfaces shall support the functional behavior described herein.

32 The system may provide additional functions or facilities not required by this c
33 standard. Nonstandard extensions should be identified as such in the system c
34 documentation. Nonstandard extensions, when used, may change the behavior of c
35 functions or facilities defined by this standard. In such cases, the system c
36 documentation shall define an environment in which an application can be run c
37 with the behavior specified by the standard. In no case shall such an environment c
38 require modification of a Strictly Conforming Application. c

39 2.2.1.2 Documentation

40 A document with the following information shall be available for an implementation c
41 claiming conformance to IEEE Std 1003.1. This document shall have the same structure c
42 as this standard, with the information presented in the appropriately numbered sections. c
43 The document shall not contain information about extended facilities or capabilities c
44 outside the scope of this standard. c

45 The document shall contain a conformance statement that indicates the full name, c
46 number, and date of the standard that applies. The conformance section may also list c
47 software standards approved by ISO or any ISO member body that are available for use c
48 by a Conforming Application. Applicable characteristics where documentation is c
49 required by one of these standards, or by standards of government bodies, may also be c
50 included. c

51 The document shall describe the contents of the <limits.h> and <unistd.h> headers, c
52 stating values, the conditions under which those values may change, and the limits of c
53 such variations. c

54 The document should describe the nature of the implementation for all implementation c
55 defined features identified in this standard. c

56 The document should specify the behavior of the implementation in those sections of this c
57 standard where it is stated that implementations may vary. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 58 **2.2.2 Application Conformance** B
 59 All applications claiming conformance to this standard shall use only **Conforming** B
 60 **Languages** §2.2.3, and shall fall within one of the following categories: B
- 61 **2.2.2.1 Strictly Conforming Application** B
 62 A **Strictly Conforming Application** is an application that requires only the facilities B
 63 described in this standard and the applicable language standards. Such an application B
 64 shall accept any behavior described in this standard as implementation defined, and for B
 65 symbolic constants, shall accept any value in the range permitted by this standard. Such B
 66 applications are permitted to adapt to the availability of facilities whose availability is B
 67 indicated by the constants in `<limits.h>` §2.9 and `<unistd.h>` §2.10. B
- 68 **2.2.2.2 Conforming Application** B
 69 A **Conforming Application** is an application that uses only the facilities described in B
 70 this standard and approved **Conforming Language** bindings for any ANSI standard. B
 71 Such an application shall include in its statement of conformance all options and limit B
 72 dependencies, all other ANSI standards used, and any other applications required. B
- 73 **2.2.2.3 Conforming Application Using Extensions** B
 74 A **Conforming Application Using Extensions** is an application that differs from a B
 75 **Conforming Application** only in that it uses non-standard facilities which are consistent B
 76 with this standard. Such an application shall fully document its requirements for these B
 77 extended facilities, in addition to the documentation required of a **Conforming** B
 78 **Application**. B
- 79 **2.2.3 Language Conformance** B
 80 As of this version of IEEE Std 1003.1, the standard has been described only in terms of C
 81 the “C” programming language. In the future, it is expected that language bindings for B
 82 other programming languages will be described as well. B
- 83 **2.2.3.1 C Language Binding** B
 84 The *ANSI/X3.159-198x Programming Language C Standard* will be used as a basis for a C
 85 C language binding to IEEE Std 1003.1. Included in the ANSI standard are definitions of B
 86 C library functions that will be required upon its final adoption. Any C language B
 87 implementation providing the facilities listed in chapter 8 of this standard shall be B
 88 deemed conforming, provided that the implementation clearly states that its C language B
 89 does not conform to *ANSI/X3.159-198x Programming Language C Standard* and its C B
 90 implementation acts only as an interim binding. B
- 91 The following rules apply to the usage of C language library functions; each of the B
 92 statements in this section applies to the detailed function descriptions in Chapters 3 B
 93 through 9, unless explicitly stated otherwise: B
- 94 If the argument to a function has an invalid value (such as a value outside the B
 95 domain of the function, or a pointer outside the address space of the program, or a B
 96 NULL pointer), the behavior is undefined. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

- 97 Any function declared in a header may also be implemented as a macro defined in B
 98 the header, so a library function should not be declared explicitly if its header is B
 99 included. B
- 100 An application may use `#undef` to remove any macro's definition to insure that B
 101 an actual function is referenced. B
- 102 Any invocation of a library function that is implemented as a macro shall expand B
 103 to code that evaluates each of its arguments only once, fully protected by B
 104 parentheses where necessary, so it is generally safe to use arbitrary expressions as B
 105 arguments. B
- 106 Provided that a library function can be declared without reference to any type B
 107 defined in a header, it is also permissible to declare the function, either explicitly B
 108 or implicitly, and use it without including its associated header. B
- 109 If a function that accepts a variable number of arguments is not declared B
 110 (explicitly, or by including its associated header), the behavior is undefined. B
- 111 **2.3 General Terms**
- 112 The following terms are used in this standard:
- 113 C
- 114 **access mode** B
 115 An access mode is a form of access permitted to a file. Each implementation shall B
 116 provide separate read, write, and execute/search access modes. B
- 117 **address space**
 118 The range of memory locations that can be referenced by a process. A
- 119 **appropriate privileges** B
 120 Each implementation shall provide a means of associating privileges with a B
 121 process with regard to the function calls and function call options defined in this B
 122 standard that need special privileges. B
- 123 **background process** C
 124 A process that is not in the (non-zero) distinguished process group of its C
 125 controlling terminal. See Job Access Control §7.1.1.5. C
- 126 **block special file** C
 127 A file that refers to a device. A block special file is normally distinguished from a C
 128 character special file by providing a more structured interface to the device. C
- 129 **character special file** C
 130 A file that refers to a device. A character special file has no defined structure and C
 131 its use is implementation defined. C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

132	child process	8
133	See process .	8
134	clock tick	8
135	A rate used within the system for scheduling and accounting. The rate is defined	8
136	by {CLK_TCK}, which is the number of intervals per second.	8
137	controlling process	B
138	The process group leader that established the connection to the controlling	B
139	terminal .	B
140	controlling terminal	8
141	A terminal that is associated with a process group . Certain input sequences from	8
142	the controlling terminal (see General Terminal Interface §7.1) cause signals to	8
143	be sent to all processes in the process group associated with the controlling	8
144	terminal .	8
145	current working directory	9
146	See working directory .	9
147	device	9
148	A computer peripheral or an object that appears to the application as such.	9
149	directory	8
150	A directory is a file that contains directory entries . No two directory entries in	8
151	the same directory shall have the same name.	C
152	directory entry (or link)	8
153	An object that associates a filename with a file . Several directory entries can	8
154	associate names with the same file .	8
155	dot	9
156	The filename consisting of a single dot character (.). See pathname resolution	9
157	§2.4.	9
158	dot-dot	9
159	The filename consisting solely of two dot characters (..). See pathname	9
160	resolution §2.4 .	9
161	effective group ID	8
162	An attribute of a process that is used in determining file access permissions (see	8
163	file access permissions §2.4). See group ID . This value is subject to change	8
164	during the process lifetime , as described in <i>setgid()</i> §4.2.2 and <i>exec</i> §3.1.2.	8
165	effective user ID	8
166	An attribute of a process that is used in determining file access permissions (see	8
167	file access permissions §2.4). See user ID . This value is subject to change during	8
168	the process lifetime , as described in <i>setuid()</i> §4.2.2 and <i>exec</i> §3.1.2.	8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

169	Epoch		C
170		The Epoch refers to the time at 0 hours, 0 minutes, 0 seconds, Coordinated	C
171		Universal Time on January 1, 1970. The value seconds since the Epoch refers to	C
172		the difference in seconds between the referenced time and the Epoch, not counting	C
173		leap seconds.	C
174	FIFO special file (or FIFO)		8
175		A FIFO special file is a file. Data written to a FIFO special file is read on a first-	8
176		in-first-out basis. Other characteristics of FIFOs are described under <i>open()</i>	9
177		§5.3.1, <i>read()</i> §6.4.1, <i>write()</i> §6.4.2, and <i>lseek()</i> §6.5.3.	9
178	file		8
179		An object that can be written to and/or read from. A file has certain attributes,	8
180		including access permissions and type. File types include regular file, character	8
181		special file, block special file, FIFO special file, and directory. Other types of	8
182		files may be defined by the implementation.	8
183	file descriptor		8
184		A file descriptor is a per-process unique, non-negative integer used to identify a	8
185		file for the purpose of file access.	8
186	file group class		C
187		A process is in the file group class of a file if the process is not in the file owner	C
188		class and if the effective group ID or one of the supplementary group IDs of the	C
189		process matches the group ID associated with the file. Other members may be	C
190		implementation defined.	C
191	file mode		B
192		The file mode contains the file permission bits and other characteristics of the file,	C
193		as described in <code><sys/stat.h></code> §5.6.1.	C
194	filename		8
195		Names consisting of 1 to {NAME_MAX} bytes may be used to name a file. The	C
196		characters composing the name may be selected from the set of all character-values	C
197		excluding the slash character and those containing the null byte (octal zero). The	9
198		filenames dot and dot-dot have special meaning; see pathname resolution §2.4.	8
199		A filename is sometimes referred to as a pathname component.	8
200	file offset		C
201		The file offset specifies the position in the file where the next I/O operation begins.	C
202		Each open file description associated with a regular file or special file has a file	C
203		offset. There is no file offset specified for a pipe or FIFO.	C
204	file other class		C
205		A process is in the file other class if the process is not in the file owner class or	C
206		file group class.	C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

207	file owner class	C
208	A process is in the file owner class of a file if the effective user ID of the process	C
209	matches the user ID of the file .	C
210	file permission bits	A
211	The file permission bits are used, along with other information, to determine if a	C
212	process has read, write, or execute/search permission to a file . The bits are divided	C
213	into three parts: owner , group , and other . Each part is used with the corresponding	C
214	file class of processes . These bits are contained in the file mode , as described in	C
215	<sys/stat.h> §5.6.1. The detailed usage of the file permission bits in access	C
216	decisions is described in file access permissions §2.4.	C
217	file serial number	8
218	A file serial number is a per-file system unique identifier for a file . File serial	8
219	numbers are not necessarily unique throughout the system.	8
220	file system	8
221	A collection of files and certain of their attributes. It provides a name space for file	9
222	serial numbers referring to those files .	9
223	foreground process	C
224	A process that is in the (non-zero) distinguished process group of its controlling	C
225	terminal . See Job Access Control §7.1.1.5.	C
226	group ID	8
227	Each system user is a member of at least one group . A group is identified by an	8
228	integer known as a group ID , which must be between zero and {UID_MAX},	8
229	inclusive. When the identity of a group is associated with a process , a group ID	8
230	value is referred to as a real group ID , an effective group ID , one of the (optional)	C
231	supplementary group IDs , or an (optional) saved set-group-ID .	8
232	Job Control Option	8
233	Job control allows users to selectively stop (suspend) the execution of processes	8
234	and continue (resume) their execution at a later point. The user typically employs	8
235	this facility via the interactive interface jointly supplied by the terminal I/O driver	8
236	and a command interpreter. Conforming implementations may optionally	8
237	support job control facilities; the presence of this option is indicated to the	C
238	application at compile time or run time by the definition of the	C
239	{_POSIX_JOB_CONTROL} symbol; see Symbolic Constants §2.10). Portions of	8
240	the standard operating system interface that are required only on implementations	8
241	that support the Job Control Option are so labeled.	8
242	job control process group leader	8
243	A job control process group leader is a process that called the <i>jcsetpgrp()</i>	8
244	function to become a process group leader . Job control process group leaders	8
245	can exist on implementations that support the Job Control Option .	8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

246	As contrasted with a session process group leader, a job control process group	8
247	leader is one of a set of processes all belonging to the same process group that	8
248	are typically controlled as a unit via the Job Control Option signaling	8
249	mechanisms. While there is usually only one session process group leader per	8
250	login session, there are usually many job control process group leaders. Side	8
251	effects typically associated with login session creation and destruction that are	8
252	performed for session process group leaders (such as effecting terminal	8
253	affiliation) are not performed for job control process group leaders.	8
254	link	8
255	See directory entry.	8
256	link count	9
257	The link count of a file is the number of directory entries that refer to that file.	9
258	mode	8
259	The mode of a file is a collection of attributes that specifies the file's type and its	8
260	access permissions. (See file access permissions §2.4).	8
261	open file	8
262	A file that is currently associated with a file descriptor.	8
263	open file description	C
264	An open file description records how a process or group of processes are	C
265	accessing a file. Each file descriptor refers to exactly one open file description,	C
266	but an open file description can be referred to by more than one file descriptor.	C
267	A file offset, file status §6.5.1.2.5, and file access modes §6.5.1.2.6 are attributes	C
268	of an open file description.	C
269	parent directory	A
270	A directory is known as a parent directory of all files that are referenced by its	A
271	directory entries, with the exception of the directory entries for dot and dot-dot.	A
272	parent process	
273	See process.	
274	parent process ID	
275	A new process is created by a currently active process. The parent process ID of	
276	a process is the process ID of its creator, for the lifetime of the creator. After the	
277	creator's lifetime has ended, the parent process ID is the process ID of an	
278	implementation defined process.	
279	path prefix	
280	A path prefix is a pathname, with an optional ending slash, that refers to a	9
281	directory.	9
282	pathname	8
283	A pathname is a string that is used to identify a file. It consists of, at most,	8

284	{PATH_MAX} bytes, including the terminating null character. It has an optional	B
285	beginning slash, followed by zero or more filenames separated by slashes.	8
286	Multiple successive slashes are considered the same as one slash. The	9
287	interpretation of the pathname is described under pathname resolution §2.4.	9
288	pathname component	C
289	See filename.	C
290	pipe	8
291	A pipe is an unnamed object created by the <i>pipe()</i> , <i>dup()</i> or <i>fcntl()</i> functions that	8
292	behaves identically to a FIFO special file for input and output.	8
293	portable filename character set	
294	The following set of graphical characters shall be portable across conforming	C
295	implementations of IEEE Std 1003.1:	C
296	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	
297	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	
298	0 1 2 3 4 5 6 7 8 9 . _ -	C
299	The last three characters are the dot, underscore, and hyphen characters,	C
300	respectively. The hyphen should not be used as the first character of a portable	C
301	filename.	C
302	privilege	B
303	See appropriate privileges.	B
304	process	8
305	An address space and single thread of control that executes within that address	8
306	space , and its required system resources. A process is created by another process	8
307	issuing the <i>fork()</i> function. The process that issues <i>fork()</i> is known as the parent	8
308	process , and the new process created by the <i>fork()</i> as the child process.	8
309	process ID	
310	Each active process in the system is uniquely identified during its lifetime by a	
311	positive integer less than or equal to {PID_MAX} called a process ID. A process	
312	ID may be re-used by the system after the process lifetime ends, provided the	
313	process was not a process group leader. If a process group leader's lifetime	
314	ends, its process ID shall not be re-used until all processes in the process group	
315	terminate.	
316	process group	8
317	Each active process is a member of a process group that is identified by a process	8
318	group ID. A newly created process joins the process group of which its creator is	8
319	a member.	8
320	process group ID	
321	The process group ID is the process ID of the initial process group leader.	

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

322	process group leader	8
323	A process group leader is a process whose process ID is the same as its process	8
324	group ID. Any process that is not a process group leader may detach itself from	8
325	its process group and become the process group leader of a new process group	8
326	by calling either the <i>setgrp()</i> or the <i>jcsetgrp()</i> function, which can cause a	8
327	process to become either a session process group leader or a job control process	8
328	group leader, respectively. Job control process group leaders can exist on	8
329	implementations that support the Job Control Option.	8
330	process lifetime	
331	After a process is created with a <i>fork()</i> function, it is considered active. Its thread	
332	of control and address space exist until it terminates. It then enters an inactive	
333	state where certain resources may be returned to the system, although some	
334	resources, such as the process ID are still in use. When another process executes a	A
335	<i>wait()</i> or <i>wait2()</i> function for an inactive process, the remaining resources are	8
336	returned to the system. The last resource to be returned to the system is the	
337	process ID. At this time, the lifetime of the process ends.	
338	read-only file system	9
339	An implementation defined characteristic of a file system that restricts file system	9
340	modifications.	9
341	real group ID	8
342	The attribute of a process that, at the time of process creation, identifies the group	8
343	of the user who created the process. See group ID. This value is subject to	8
344	change during the process lifetime, as described in <i>setgid()</i> §4.2.2.	8
345	real user ID	8
346	The attribute of a process that, at the time of process creation, identifies the user	8
347	who created the process. See user ID. This value is subject to change during the	8
348	process lifetime, as described in <i>setuid()</i> §4.2.2.	8
349	regular file	8
350	A file that is a randomly accessible sequence of bytes, with no further structure	A
351	imposed by the system.	A
352	root directory	9
353	A directory, associated with a process, that is used in pathname resolution §2.4	9
354	for pathnames that begin with a slash.	9
355		B
356	saved set-group-ID	8
357	When the saved set-group-ID option is implemented, the saved set-group-ID is	8
358	an attribute of a process that allows some flexibility in the assignment of the	8
359	effective group ID attribute, as described in <i>setgid()</i> §4.2.2, and <i>exec</i> §3.1.2.	8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

360	saved set-user-ID	8
361	When the saved set-user-ID option is implemented, the saved set-user-ID is an	8
362	attribute of a process that allows some flexibility in the assignment of the effective	8
363	user ID attribute, as described in <i>setuid()</i> §4.2.2, and <i>exec</i> §3.1.2.	8
364	session process group leader	8
365	A session process group leader is a process that called the <i>setpgrp()</i> function to	8
366	become a process group leader. When the Job Control Option is not	8
367	implemented, this term is a synonym for process group leader. When the Job	8
368	Control Option is implemented, this term is used to distinguish the functionality	8
369	of the <i>setpgrp()</i> function from that of the <i>jcsetpgrp()</i> function, which establishes a	8
370	job control process group leader.	8
371	As contrasted with a job control process group leader, there is typically only one	8
372	session process group leader per login session and it is the main command	8
373	interpreter for the session. All processes created during the session are	8
374	descendants of the session process group leader and members of the same	8
375	process group.	8
376	signal	9
377	A mechanism by which a process may be notified of, or affected by, an event	9
378	occurring in the system. Examples of such events include hardware exceptions and	9
379	specific actions by processes. The term signal is also used to refer to the event	9
380	itself.	9
381	slash	A
382	The term slash is used to represent the literal character “/”. This character is also	C
383	known as “solidus” in ISO DIS 8895/1.	C
384		B
385	supplementary group ID	
386	A process has up to {NGROUPS_MAX} supplementary group IDs used in	
387	determining file access permissions, in addition to the effective group ID. The	8
388	supplementary group IDs of a process are set to the supplementary group IDs	8
389	of the parent process when the process is created.	8
390	system	8
391	The term system is used in this standard to refer to an implementation of this	8
392	standard.	8
393	system process	C
394	A process that runs on behalf of the system. It may have special implementation	C
395	defined characteristics.	C
396	terminal (or terminal device)	8
397	A character special file that obeys the specifications of the General Terminal	9
398	Interface §7.1.	9

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

399	terminal group ID	C
400	The attribute of a process that is used to identify the controlling terminal for a	B
401	login session. All processes in a process group that have a controlling terminal	B
402	share the same controlling terminal. That is, the terminal group ID is either	C
403	cleared or has the same value for all processes in a process group.	C
404	user ID	8
405	Each system user is identified by an integer known as a user ID, which must be	8
406	between zero and {UID_MAX}, inclusive. When the identity of a user is	8
407	associated with a process, a user ID value is referred to as a real user ID, an	8
408	effective user ID, or an (optional) saved set-user-ID.	8
409	working directory (or current working directory)	9
410	A directory, associated with a process, that is used in pathname resolution §2.4	9
411	for pathnames that do not begin with a slash.	9
412	2.4 General Concepts	
413	file access permissions	C
414	File access control is provided using the file permission bits along with	C
415	other information. These bits are set at file creation, <i>open()</i> §5.3.1 or	C
416	<i>creat()</i> §5.3.2, and are changed by <i>chmod()</i> §5.6.4. These bits are read by	C
417	<i>stat()</i> or <i>fstat()</i> §5.6.2.	C
418	Whenever a process requests file access permission for read, write, or	C
419	execute/search, the following applies:	C
420	If the process has appropriate privileges to override the access	C
421	mechanism:	C
422	If read, write, or directory search is requested, access is	C
423	granted.	C
424	If execute permission is requested, access is granted if at	C
425	least one of the execute file permission bits is set, or if an	C
426	implementation defined access mechanism is enabled that	C
427	allows execute permission; otherwise, access is denied.	C
428	Otherwise, the access mechanism is:	C
429	If the requested access permission bit is set in the part	C
430	(owner/group/other) of the file permission bits that	C
431	corresponds to the file class (owner/group/other) of the	C
432	process, or if an implementation defined access mechanism	C
433	is enabled that allows the requested permission, access is	C
434	granted, unless the process is denied access by an	C
435	implementation defined constraint.	C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

436	Otherwise, access is denied.	C
437	An implementation may provide an alternative access mechanism,	C
438	enabled explicitly by the user, that does not necessarily use the file	C
439	permission bits. This alternative access mechanism shall:	C
440	• Specify appropriate file permission bits for the owner, group,	C
441	and other classes of the file to be returned by <i>stat()</i> or <i>fstat()</i> .	C
442	• Be enabled only by explicit user action.	C
443	• Be disabled after the file permission bits are changed by	C
444	<i>chmod()</i> .	C
445	file hierarchy	9
446	Files in the system are organized in a hierarchical structure in which all of	9
447	the non-terminal nodes are directories and all of the terminal nodes are	C
448	any other type of file. Because multiple directory entries may refer to	9
449	the same file, the hierarchy is properly described as a <i>directed graph</i> .	
450	filename portability	
451	Filenames should be constructed from the portable filename character	
452	set because the use of other characters can be confusing or ambiguous in	
453	certain contexts.	
454	file times update	C
455	Each file has three associated time values that are updated when file data	C
456	has been accessed, file data has been modified, or file status has been	C
457	changed, respectively. These values are returned in the file characteristics	C
458	structure, as described in <code><sys/stat.h></code> §5.6.1.	C
459	For each function in this standard that reads or writes file data or changes	C
460	the file status, the appropriate time-related fields are noted as “marked-	C
461	for-update.” At an update point in time, any marked fields are set to the	C
462	current time and the update marks are cleared. One such update point is	C
463	when the file is no longer open by any process. Additional update points	C
464	are implementation defined. Updates are not done for files on read-only	C
465	file systems.	C
466	pathname resolution	9
467	Pathname resolution is performed for a process to resolve a pathname	9
468	to a particular file in a file hierarchy. There may be multiple pathnames	9
469	that resolve to the same file.	9
470	Each filename in the pathname is located in the directory specified by its	9
471	predecessor (for example, in the pathname fragment “a/b”, file “b” is	9
472	located in directory “a”). Pathname resolution fails if this cannot be	9
473	accomplished. If the pathname begins with a slash, the predecessor of	9

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

474 the first filename in the **pathname** is taken to be the root directory of the 9
 475 process (such **pathnames** are referred to as **absolute pathnames**). If the 9
 476 **pathname** does not begin with a slash, the predecessor of the first 9
 477 filename of the **pathname** is taken to be the current working directory 9
 478 of the process (such **pathnames** are referred to as **relative pathnames**). 9

479 The interpretation of a **pathname** component is dependent on the values c
 480 of `{NAME_MAX}` and `{_POSIX_NO_TRUNC}` associated with the path c
 481 prefix of that component. If any **pathname** component is longer than c
 482 `{NAME_MAX}`, and `{_POSIX_NO_TRUNC}` is in effect for the path c
 483 prefix of that component (see *pathconf()* §5.7.1), the implementation c
 484 shall consider this an error condition. Otherwise, the implementation shall c
 485 use the first `{NAME_MAX}` bytes of the **pathname** component. c

486 The special filename, **dot**, refers to the directory specified by its 9
 487 predecessor. The special filename, **dot-dot**, refers to the parent 9
 488 directory of its predecessor directory. As a special case, in the root 9
 489 directory, **dot-dot** may refer to the root directory itself. 9

490 A **pathname** consisting of a single slash resolves to the root directory of 9
 491 the process. If `{_POSIX_PATHNAME_NULL}` is defined, a null c
 492 **pathname** (a **pathname** consisting of a null string) resolves to the c
 493 current working directory of the process; otherwise, a null **pathname** is c
 494 invalid. c
 495 9

496 2.5 Error Numbers

497 Most functions provide an error number in the external variable *errno*, which is defined 9
 498 as: 9

```
499     extern int errno; 9
```

500 This variable is defined only after calls to functions for which it is explicitly stated to be B
 501 set. The variable *errno* should only be examined when it is indicated to be valid by a
 502 function's return value. No function defined in this standard sets *errno* to zero to indicate B
 503 an error.

504 If more than one error occurs in processing a function call, this standard does not define
 505 in what order the errors are detected; therefore, any one of the possible errors may be
 506 returned.

507 Implementations may support additional errors not included in this list, may generate
 508 errors included in this list under circumstances other than those described here, or may
 509 contain extensions or limitations that prevent some errors from occurring. The Errors 9
 510 subsection in each function description specifies which error conditions shall be required 9

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

511		and which may be implementation defined. Implementations shall not generate an error	
512		number different from the ones described here for error conditions described in this	
513		standard.	
514		The following symbolic names identify the possible error numbers, in the context of	8
515		functions specifically defined in this standard; these general descriptions are more	B
516		precisely defined in the Errors sections of functions that return them. Only these	
517		symbolic names should be used in programs, since the actual value of the error number is	
518		implementation defined. All values shall be unique numbers. The implementation	B
519		defined values for these names can be found in the header <code><errno.h></code> .	
520	[E2BIG]	Arg list too long	
521		The sum of the number of bytes used by the new process image's	
522		argument list and environment list is greater than the system-	
523		imposed limit of {ARG_MAX} bytes.	
524	[EACCES]	Permission denied	
525		An attempt was made to access a file in a way forbidden by its file	9
526		access permissions.	8
527	[EAGAIN]	Resource temporarily unavailable	8
528		This is a temporary condition and later calls to the same routine	8
529		may complete normally.	8
530			A
531	[EBADF]	Bad file number	8
532		A file descriptor argument is out of range, refers to no open file, or	8
533		a read (write) request is made to a file that is only open for writing	8
534		(reading).	8
535	[EBUSY]	Resource busy	8
536		An attempt was made to make use of a system resource that is not	8
537		currently available because it is being used by another process in a	8
538		manner that would conflict with the request being made by this	8
539		process.	8
540	[ECHILD]	No child processes	8
541		A <code>wait()</code> or <code>wait2()</code> function was executed by a process that had	8
542		no existing or unwaited-for child processes.	
543	[EDEADLK]	Resource deadlock would occur	
544		A process that has locked a system resource would have been put	
545		to sleep while attempting to access a resource locked by another	
546		process.	
547	[EDOM]	Domain error	B
548		Defined in <i>ANSI/X3.159-198x Programming Language C</i>	B
549		<i>Standard</i> ; an input argument is outside the defined domain of the	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

550		mathematical function.	B
551	[EEXIST]	File exists	
552		An existing file was mentioned in an inappropriate context, for	
553		instance, as the new link name in a <i>link()</i> function.	
554	[EFAULT]	Bad address	
555		The system detected an invalid address in attempting to use an	
556		argument of a call. The reliable detection of this error is	
557		implementation defined; however, implementations that do detect	B
558		this condition shall use this value.	B
559	[EFBIG]	File too large	
560		The size of a file would exceed an implementation defined	C
561		maximum file size.	C
562	[EINTR]	Interrupted function call	C
563		An asynchronous signal (such as SIGINT or SIGQUIT; see the	
564		description of header <signal.h> §3.3.1) was caught by the process	C
565		during the execution of an interruptible function. If the signal	C
566		handler performs a normal return, the interrupted function call may	C
567		return this error condition.	C
568	[EINVAL]	Invalid argument	
569		Some invalid argument (for example, mentioning an undefined	
570		signal in a <i>signal()</i> function or a <i>kill()</i> function).	
571	[EIO]	Input/output error	
572		Some physical input or output error has occurred. This error may	8
573		be reported on a subsequent operation on the same file descriptor.	8
574		Any other error-causing operation on the same file descriptor may	8
575		cause the [EIO] error indication to be lost.	8
576	[EISDIR]	Is a directory	
577		An attempt was made to open a directory with write mode	
578		specified.	
579	[EMFILE]	Too many open files	
580		An attempt was made to open more than the maximum number of	
581		{OPEN_MAX} file descriptors allowed in this process.	
582	[EMLINK]	Too many links	B
583		An attempt was made to have the link count of a single file exceed	8
584		{LINK_MAX}.	8
585	[ENAMETOOLONG]	Filename too long	
586		The size of a pathname string exceeds {PATH_MAX}, or a	C
587		pathname component is longer than {NAME_MAX} while	C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

588		{_POSIX_NO_TRUNC} is in effect.	C
589	[ENFILE]	Too many open files in system	
590		Too many files are currently open in the system. The system has	B
591		reached its predefined limit for simultaneously open files and	B
592		temporarily cannot accept requests to open another one.	
593	[ENODEV]	No such device	
594		An attempt was made to apply an inappropriate function to a	
595		device; for example, trying to read a write-only device such as a	
596		printer.	
597	[ENOENT]	No such file or directory	
598		A component of a specified pathname does not exist, or the	A
599		pathname is an empty string.	A
600	[ENOEXEC]	Exec format error	
601		A request is made to execute a file that, although it has the	
602		appropriate permissions, is not in the proper format.	A
603	[ENOLCK]	No locks available.	B
604		A system-imposed limit on the number of simultaneous file and	B
605		record locks has been reached and no more are currently available.	B
606	[ENOMEM]	Not enough space	
607		The new process image requires more memory than is allowed by	
608		the hardware or system-imposed memory management constraints.	
609	[ENOSPC]	No space left on device	
610		During a <i>write()</i> function on a regular file or when extending a	
611		directory, there is no free space left on the device.	
612	[ENOTDIR]	Not a directory	
613		A component of the specified pathname exists, but it is not a	B
614		directory, when a directory was expected.	B
615	[ENOTEMPTY]	Directory not empty	B
616		A directory with entries other than dot and dot-dot was supplied	B
617		when an empty directory was expected.	B
618	[ENOTTY]	Inappropriate I/O control operation	A
619		A control function has been attempted for a file or special file for	A
620		which the operation is inappropriate.	A
621	[ENXIO]	No such device or address	
622		Input or output on a special file refers to a device that does not	8
623		exist, or makes a request beyond the limits of the device. It may	8
624		also occur when, for example, a tape drive is not on-line or no disk	8
625		pack is loaded on a drive.	8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

626	[E P ERM]	Operation not permitted	8
627		An attempt was made to perform an operation limited to processes	B
628		with appropriate privileges or to the owner of a file or other	B
629		resource.	B
630	[EPIPE]	Broken pipe	8
631		A write on a pipe or FIFO for which there is no process to read the	8
632		data. This condition normally generates the signal SIGPIPE; the	8
633		error is returned if the signal is ignored.	8
634	[ERANGE]	Result too large	8
635		Defined in <i>ANSI/X3.159-198x Programming Language C</i>	C
636		<i>Standard</i> ; the result of the function is too large to fit in the	8
637		available space.	8
638	[EROFS]	Read only file system	
639		An attempt was made to modify a file or directory on a file system	
640		that is read only.	
641	[ESPIPE]	Invalid seek	
642		An <i>lseek()</i> function was issued on a pipe or FIFO.	
643	[ESRCH]	No such process	
644		No process can be found corresponding to that specified by the	
645		given process ID.	
646	[EXDEV]	Improper link	
647		A link to a file on another file system was attempted.	

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

648 **2.6 Primitive System Data Types**

649 Some data types used by the various system functions are not defined as part of this
 650 standard, but are defined by the implementation. These types are then defined in the
 651 header `<sys/types.h>`, which contains definitions for at least the following types:

<u>Defined Type</u>	<u>Description</u>	
<code>clock_t</code>	Used for system times (in {CLK_TCK}ths of a second)	8 8
<code>dev_t</code>	Used for device numbers	
<code>ino_t</code>	Used for file serial numbers	
<code>mode_t</code>	Used for some file attributes, e.g. file type, file access permissions	9 9
<code>nlink_t</code>	Used for link counts	9
<code>off_t</code>	Used for file sizes	
<code>time_t</code>	Used for system times (in seconds)	
<code>uid_t</code>	Used for user IDs and group IDs	9

660 All of the types listed above shall be integral types. B

661 Additional type definitions may also be given in this header. C

662 **2.7 Environment Description**

663 An array of strings called the environment is made available when a process begins. This
 664 array is pointed to by the external variable `environ`, which is defined as:

665 **extern char **environ;**

666 These strings have the form "`name=value`". There is no meaning associated with the C
 667 order of the strings in the environment. If more than one string in a process's A
 668 environment has the same `name`, the consequences are undefined. The following names
 669 may be defined and have the indicated meaning if they are defined:

670	HOME	Name of the user's initial working directory, from the	
671		password database (see description of the header <code><pwd.h></code>	
672		§9.2.2).	
673	IFS	Characters used as field separators. The format of this string is	A
674		currently not defined as part of this standard.	A
675	LANG	Specifies the name of the pre-defined setting for locale.	C
676	LC_CTYPE	Specifies the name of the locale for character classification.	C
677	LC_COLLATE	Specifies the name of the locale for collation information.	C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

678	LC_TIME	Specifies the name of the locale for date/time formatting information.	C
679			C
680	LC_NUMERIC	Specifies the name of the locale containing numeric editing (i.e., radix character) information.	C
681			C
682	LOGNAME	The name of the user's login account, corresponding to the login name in the password database (see description of the header <pwd.h>).	
683			
684			
685	MAIL	System mailer information. The format of this string is currently not defined as part of this standard.	A
686			A
687	PATH	The sequence of path prefixes that certain commands and functions apply in searching for a file known by an incomplete pathname (a pathname without a leading slash). The prefixes are separated by a colon (:). When a non-zero-length prefix is applied to an incomplete pathname, a slash is inserted between the prefix and the incomplete pathname. A zero-length prefix is a special prefix that indicates the current working directory. It appears as two adjacent colons ("::"), as an initial colon preceding the rest of the list, or as a trailing colon following the rest of the list. The list is searched from left to right until an executable program by the specified name is found. If the filename being sought contains a slash, the search through path prefixes is not done.	9
688			9
689			9
690			9
691			9
692			9
693			9
694			9
695			9
696			9
697			9
698			9
699			9
700	PS1	Prompting string for interactive programs. The format of this string is currently not defined as part of this standard.	A
701			A
702	PS2	Prompting string for interactive programs. The format of this string is currently not defined as part of this standard.	A
703			A
704	SHELL	The shell command interpreter name. The format of this string is currently not defined as part of this standard.	A
705			A
706	TERM	The terminal type for which output is to be prepared. This information is used by commands and application programs wishing to exploit special capabilities specific to a terminal.	
707			
708			
709	TZ	Time zone information. The format of this string is defined in <i>asctime()</i> §8.1.1.	C
710			C
711	It is recommended that the environment variable <i>names</i> consist solely of characters from the portable filename character set. Other valid characters may be permitted by an implementation, but use of them by an application may limit its portability. Upper- and lowercase letters retain their unique identities and are not folded together. It is recommended that only capital letters, underscores, and numbers be used for		
712			
713			
714			
715			

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

716 environment variable names and that the first character be a letter.

717 The *values* that the environment variables may be assigned are not restricted except that
 718 they are considered to end with a null byte and the total space used to store the
 719 environment and the arguments to the process is limited to {ARG_MAX} bytes.

720 Other *name=value* pairs may be placed in the environment by manipulating the *environ*
 721 variable or by using *envp* arguments when creating a process (see *exec* §3.1.2).

722 A

723 2.8 C Language Definitions

724 Certain terms used in this standard are considered to be defined by the C programming 8
 725 language. The following terms are defined in the *ANSI/X3.159-198x Programming* 8
 726 *Language C Standard* (see C Language Standard §A.2.1): 8

727 **NULL** 8

728 **byte** C

729 **character**

730 **character array**

731 **string**

732 **empty string**

733 The term **NULL pointer** in this standard is equivalent to the term **null pointer** used in
 734 the *ANSI/X3.159-198x Programming Language C Standard*.

735 2.9 Numerical Limits

736 The following subsections list magnitude limitations imposed by a specific A
 737 implementation. A standard conforming implementation shall define each of the values A
 738 specified below as a symbolic constant in the header `<limits.h>`. The values given below A
 739 shall be replaced by restricted constant expressions suitable for use in `#if` preprocessing A
 740 directives. The braces notation, {LIMIT}, is used in the standard to indicate these values, A
 741 but the braces are not part of the name. A

742 **2.9.1 C Language Limits**

743 Certain limits used in this standard are considered to be defined in the C programming
 744 language. The following limits are defined in the *ANSI/X3.159-198x Programming*
 745 *Language C Standard* (see C Language Standard §A.2.1):

746	CHAR_BIT	A
747	CHAR_MAX	A
748	CHAR_MIN	A
749	CLK_TCK	C
750	INT_MAX	A
751	INT_MIN	A
752	LONG_MAX	A
753	LONG_MIN	A
754	SCHAR_MAX	A
755	SCHAR_MIN	A
756	SHRT_MAX	A
757	SHRT_MIN	A
758	UCHAR_MAX	A
759	UINT_MAX	A
760	ULONG_MAX	A
761	USHRT_MAX	A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

762 2.9.2 Run-Time Invariant Values

763 The following magnitude limitations shall be fixed for a specific implementation. A
 764 **Strictly Conforming Application** shall assume that the value supplied by <limits.h> in
 765 a specific implementation is that which pertains whenever the **Strictly Conforming**
 766 **Application** is run under that implementation. A specific instance of a specific
 767 implementation shall not vary the value from that supplied by <limits.h> for that
 768 implementation.

<u>Name</u>	<u>Description</u>	<u>Minimum Value</u>	
MAX_INPUT	Maximum number of bytes allowed in a terminal input queue	256	A C C
NGROUPS_MAX	Maximum number of simultaneous supplementary group IDs per process	0	A A A
PASS_MAX	Maximum number of bytes in a password (not a string length; does not include a terminating null)	8	B B B B
PID_MAX	Maximum value for a process ID	30000	A
UID_MAX	Maximum value for a user or group ID	32000	A A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

795 2.9.3 Run-Time Invariant Values (Possibly Indeterminate) C

796 A definition of one of the following values shall be omitted from the `<limits.h>` on A
 797 specific implementations where the corresponding value is equal to or greater than the A
 798 stated minimum, but is indeterminate. This depends, for example, on the amount of A
 799 available memory space on a specific instance of a specific implementation. A

<u>Name</u>	<u>Description</u>	<u>Minimum Value</u>	
ARG_MAX	Maximum length of arguments for <i>exec()</i> in bytes, including <i>environ</i> data	4096	A B B B
CHILD_MAX	Maximum number of simultaneous processes per user ID	6	C C C
MAX_CANON	Maximum number of bytes in a terminal canonical input line. (See Canonical Mode Input Processing §7.1.1.7.)	256	B B B B
OPEN_MAX	Maximum number of files that one process can have open at any given time	16	C C C

815 2.9.4 Pathname Variable Values C

816 The following values may be constants within an implementation, or may vary from one C
 817 pathname to another. For example, file systems or directories may have different C
 818 characteristics. C

819 A definition of one of the following values shall be omitted from the `<limits.h>` on C
 820 specific implementations where the corresponding value is equal to or greater than the C
 821 stated minimum, but is indeterminate. The actual value supported for a specific C
 822 pathname shall be provided by the *pathconf()* §5.7.1 function. C

<u>Name</u>	<u>Description</u>	<u>Minimum Value</u>	
NAME_MAX	Maximum number of bytes in a file name (not a string length; does not include a terminating null).	14	C C C C
PATH_MAX	Maximum number of bytes in a pathname (not a string length; does not include a terminating null).	255	C C C C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

833 **2.9.5 Run-Time Inceasable Values**

834 The following magnitude limitations shall be fixed by specific implementations. A
 835 **Strictly Conforming Application** shall assume that the value supplied by `<limits.h>` in
 836 a specific implementation is the minimum that pertains whenever the **Strictly**
 837 **Conforming Application** is run under that implementation. A specific instance of a
 838 specific implementation may increase the value relative to that supplied by `<limits.h>`
 839 for that implementation.

<u>Name</u>	<u>Description</u>	<u>Minimum Value</u>	
LINK_MAX	Maximum value of a file's link count	8	c c
PIPE_BUF	Maximum number of bytes that is guaranteed to be atomic when writing to a pipe	512	c c c

845 c

846 **2.10 Symbolic Constants**

847 A conforming implementation shall have a header with the name `<unistd.h>`. This file
 848 defines the symbolic constants and structures referenced elsewhere in the standard and
 849 not already defined or declared in some other header. When used, it shall be referenced
 850 as follows:

851 `#include <unistd.h>`

852 The constants defined in this file are shown below. The actual values of the constants are
 853 implementation defined.

854 **2.10.1 Symbolic constants for the *access()* function**

<u>Constant</u>	<u>Description</u>
R_OK	Test for read Permission
W_OK	Test for write Permission
X_OK	Test for execute or search Permission
F_OK	Test for existence of file

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

860 B

866 2.10.2 Symbolic constant for the *lseek()* function

<u>Constant</u>	<u>Description</u>	
SEEK_SET	Set file offset to <i>offset</i>	c
SEEK_CUR	Set file offset to current plus <i>offset</i>	c
SEEK_END	Set file offset to EOF plus <i>offset</i>	c

871 B

872 2.10.3 Compile time symbolic constants for portability specifications c

873 These constants may be used by the application, at compile time, to determine which c
 874 optional facilities are present and what actions shall be taken by the implementation. c

875 Some of these symbols may have more liberal, or less restrictive, values at the time of c
 876 execution. Although a Strictly Conforming Application can rely on the values compiled c
 877 from the `<unistd.h>` header to afford it portability on all instances of an implementation, c
 878 it may choose to interrogate a value at run time to take advantage of the current c
 879 configuration. See *sysconf()* §4.8.1. c

880 { _POSIX_EXIT_SIGHUP } c
 881 If defined, if the process is a session process group leader, the c
 882 *_exit()* §3.2.2 function will send the SIGHUP signal to all c
 883 processes with group IDs equal to that of the calling process. c

884 { _POSIX_JOB_CONTROL } c
 885 If this symbol is defined, it indicates that the implementation c
 886 supports the Job Control Option. c

887 { _POSIX_KILL_PID_NEG1 } c
 888 If defined, a *kill()* §3.3.2 function call with *pid* of -1 will send the c
 889 signal to the sending process; otherwise, the sending process will c
 890 be excluded. c

891 { _POSIX_KILL_SAVED } c
 892 If defined, and if { _POSIX_SAVED_IDS } is also defined, the *kill()* c
 893 §3.3.2 uses the saved set-user-ID instead of the effective user-ID. c

894 { _POSIX_PATHNAME_NULL } c
 895 If defined, a null pathname resolves to the current working c
 896 directory; otherwise, a null pathname is considered invalid. c

897 { _POSIX_PGID_CLEAR } c
 898 If defined, if the process is a session process group leader, the c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

899	<code>_exit()</code>	§3.2.2 function will cause all process group IDs equal to	c
900		that of the calling process to have their process group IDs set to	c
901		zero.	c
902	<code>{_POSIX_SAVED_IDS}</code>		c
903		If defined, the <code>exec()</code> §3.1.2 saves the effective user and group IDs.	c
904	<code>{_POSIX_VERSION}</code>		c
905		The integer value 198803. This value will change with each	c
906		published version or revision of this standard to indicate the (4-	c
907		digit) year and (2-digit) month that the standard was approved by	c
908		the IEEE Standards Board. <i>Editor's Note: The value 198803 is</i>	c
909		<i>tentative as of this draft. The published Full Use Standard will</i>	c
910		<i>contain the value that should be used by applications; however, it</i>	c
911		<i>is guaranteed to not be less than 198803.</i>	c
912	2.10.4 Execution time symbolic constants for portability specifications		c
913	These constants may be used by the application, at execution time, to determine which		c
914	optional facilities are present and what actions shall be taken by the implementation in		c
915	some circumstances described by this standard as implementation defined .		c
916	If any of the following constants are not defined in the header <code><unistd.h></code> , the value		c
917	varies depending on the file to which it is applied. See <code>pathconf()</code> §5.7.1.		c
918	If any of the following are defined to have value <code>-1</code> in the header <code><unistd.h></code> , the		c
919	implementation shall not provide the option on any file. If any of the following are		c
920	defined to have a value other than <code>-1</code> in the header <code><unistd.h></code> , the implementation shall		c
921	provide the option on all applicable files.		c
922	All of the following, whether defined in <code><unistd.h></code> or not, may be queried with respect		c
923	to a specific file using the <code>pathconf()</code> or <code>fpathconf()</code> functions.		c
924	<code>{_POSIX_CHOWN_RESTRICTED}</code>		c
925		The use of the <code>chown()</code> §5.6.5 function is restricted to a process	c
926		with appropriate privileges.	c
927	<code>{_POSIX_CHOWN_SUP_GRP}</code>		c
928		The use of the <code>chown()</code> §5.6.5 function is restricted to changing	c
929		the group ID of a file only to the effective group ID of the process	c
930		or to one of its supplementary group IDs.	c
931	<code>{_POSIX_DIR_DOTS}</code>		c
932		An “empty directory” contains entries for dot and dot-dot;	c
933		otherwise it must be completely empty.	c
934	<code>{_POSIX_GROUP_PARENT}</code>		c
935		A newly created file, directory, or FIFO receives the group ID of its	c
936		parent directory; otherwise, the process's effective group ID is	c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

937	used.	c
938	{_POSIX_LINK_DIR}	c
939	Any user is allowed to <i>link()</i> §5.3.4 or <i>unlink()</i> §5.5.1 directories.	c
940	{_POSIX_NO_TRUNC}	c
941	Pathname components longer than {NAME_MAX} generate an	c
942	error.	c
943	{_POSIX_UTIME_OWNER}	c
944	The owner of a file is allowed to use the <i>utime()</i> §5.6.6 function	c
945	with a non-NULL argument.	c
946	{_POSIX_V_DISABLE}	c
947	Terminal special characters defined in <termios.h> §7.1.2 can be	c
948	disabled using this character value, if it is defined. See <i>tcgetattr()</i>	c
949	and <i>tcsetattr()</i> §7.2.1.	c

3. Process Primitives

1 The functions described in this chapter perform the most primitive operating system
2 services dealing with processes, interprocess signals, and timers. All attributes of a
3 process that are specified in this standard shall remain unchanged by a process primitive 8
4 unless the description of that process primitive states explicitly that the attribute is
5 changed.

6 3.1 Process Creation

7 Running a program takes two steps: first, the *fork()* function is called to produce a new
8 process, then that new process calls one of the *exec* functions to start the new program.

9 3.1.1 Process Creation

10 Function: *fork()*

11 3.1.1.1 Synopsis

12 `int fork ()`

13 3.1.1.2 Description

14 The *fork()* function shall cause creation of a new process. The new process (child
15 process) shall be an exact copy of the calling process (parent process) except for the
16 following:

17 The child process has a unique process ID. If the implementation supports the Job B
18 Control Option, the child process ID also does not match any active process group B
19 ID. B

20 The child process has a different parent process ID (which is the process ID of the
21 parent process).

22 The child process has its own copy of the parent's file descriptors. Each of the c
23 child's file descriptors refers to the same open file description with the c
24 corresponding file descriptor of the parent. c

25 The child process's values of *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime*
26 are set to zero (see *times()* §4.5.2).

- 27 File locks previously set by the parent are not inherited by the child. (See *fcntl()* B
 28 §6.5.2.)
- 29 Pending alarms are cleared for the child process. (See *alarm()* §3.4.1.) B
- 30 The set of signals pending for the child process is initialized to the empty set. 8
 31 (See *<signal.h>* §3.3.1.) 8
- 32 All other process characteristics defined by this standard shall be the same in the parent 8
 33 and the child processes. The inheritance of process characteristics not defined by this B
 34 standard is implementation defined and shall be documented in the system B
 35 documentation. (See Documentation §2.2.1.2.) B
- 36 If during the *fork()* function call, a signal is directed to a group of processes of which the C
 37 child process is a member, whether or not the signal is delivered to the child process is C
 38 undefined. (See *kill()* §3.3.2.) C
- 39 **3.1.1.3 Returns**
- 40 Upon successful completion, *fork()* shall return to the child process a value of zero and
 41 shall return to the parent process the process ID of the child process, and both processes
 42 shall continue to execute from the *fork()* function. Otherwise, a value of -1 shall be
 43 returned to the parent process, no child process shall be created, and *errno* shall be set to
 44 indicate the error.
- 45 **3.1.1.4 Errors**
- 46 If any of the following conditions occur, the *fork()* function shall return -1 and set *errno* B
 47 to the corresponding value: B
- 48 [EAGAIN] The system lacked the necessary resources to create another B
 49 process, or; the system-imposed limit on the total number of B
 50 processes under execution by a single user would be exceeded. B
- 51 9
- 52 For each of the following conditions, if the condition is detected, the *fork()* function shall B
 53 return -1 and set *errno* to the corresponding value: B
- 54 [ENOMEM] The process requires more space than the system is able to supply.
- 55 **3.1.1.5 References**
- 56 *alarm()* §3.4.1, *exec* §3.1.2, *fcntl()* §6.5.2, *kill()* §3.3.2, *times()* §4.5.2, *wait* §3.2.1. B

57 **3.1.2 Execute a File**58 Functions: *execl()*, *execv()*, *execle()*, *execve()*, *execlp()*, *execvp()*59 **3.1.2.1 Synopsis**

```

60         int execl (path, arg0, arg1, ..., argn, (char *) 0)
61         char *path, *arg0, *arg1, ..., *argn;

62         int execv (path, argv)
63         char *path, *argv[ ];

64         int execle (path, arg0, arg1, ..., argn, (char *) 0, envp)
65         char *path, *arg0, *arg1, ..., *argn, *envp[ ];

66         int execve (path, argv, envp);
67         char *path, *argv[ ], *envp[ ];

68         int execlp (file, arg0, arg1, ..., argn, (char *) 0)
69         char *file, *arg0, *arg1, ..., *argn;

70         int execvp (file, argv)
71         char *file, *argv[ ];

72         extern char **environ;

```

73 **3.1.2.2 Description**

74 The *exec* family of functions shall replace the current process image with a new process
75 image. The new image is constructed from a regular, executable file called the *new*
76 *process image file*. There shall be no return from a successful *exec*, because the calling **B**
77 process image is overlaid by the new process image.

78 When a C program is executed as a result of this call, it shall be entered as a C language
79 procedure call as follows:

```

80         extern char **environ;
81         int main (argc, argv)
82         int argc;
83         char **argv;

```

B

84 where *argc* is the argument count (one or greater), *argv* is an array of character pointers
85 to the arguments themselves and *environ* is an array of character pointers to the **B**
86 environment strings. The *environ* array is terminated by a NULL pointer. **C**

87 The arguments specified by a program with one of the *exec* functions shall be passed on
88 to the new process image in the corresponding *main()* arguments.

89 The argument *path* points to a pathname that identifies the new process image file.

90 The argument *file* points to the new process image file. If the *file* argument does not **B**
91 contain a slash character, the path prefix for this file is obtained by a search of the
92 directories passed as the environment variable PATH (see Environment Description

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 93 §2.7). If this environment variable is not present, the results of the search are B
 94 implementation defined. B
- 95 The arguments *arg0*, *arg1*, ..., *argn* are pointers to null-terminated character strings.
 96 These strings constitute the argument list available to the new process image. The list is
 97 terminated by a NULL pointer. The argument *arg0* should point to a filename that is
 98 associated with the process being started by one of the *exec* functions.
- 99 The argument *argv* is an array of character pointers to null-terminated strings. The last
 100 member of this array shall be a NULL pointer. These strings constitute the argument list
 101 available to the new process image. The value in *argv[0]* should point to a filename that
 102 is associated with the process being started by one of the *exec* functions.
- 103 The argument *envp* is an array of character pointers to null-terminated strings. These
 104 strings constitute the environment for the new process image. The *envp* array is
 105 terminated by a NULL pointer.
- 106 For those forms not containing an *envp* pointer (*execl()*, *execv()*, *execlp()*, and *execvp()*)
 107 the environment is taken from the external variable *environ*.
- 108 The number of bytes available for the new process's combined argument and B
 109 environment lists is {ARG_MAX}. The implementation shall specify in the system B
 110 documentation (see Documentation §2.2.1.2) whether null terminators, pointers, and/or B
 111 any alignment bytes, are included in this total. B
- 112 File descriptors open in the calling process image remain open in the new process image,
 113 except for those whose close-on-exec flag FD_CLOEXEC is set (see *fcntl()* §6.5.2, B
 114 <fcntl.h> §6.5.1). For those file descriptors that remain open, all attributes of the open C
 115 file description remain unchanged. C
- 116 The file locks held by a process are not affected by the *exec* functions. See *fcntl()* §6.5.2. B
- 117 Signals set to the default action (SIG_DFL) in the calling process image shall be set to the
 118 default action in the new process image. Signals set to be ignored (SIG_IGN) by the
 119 calling process image shall be set to be ignored by the new process image. Signals set to
 120 be caught by the calling process image shall be set to the default action in the new
 121 process image (see *sigaction()* §3.3.4). C
- 122 If the set-user-ID mode bit of the new process image file is set (see *chmod()* §5.6.4), the
 123 effective user ID of the new process image is set to the owner ID of the new process
 124 image file. Similarly, if the set-group-ID mode bit of the new process image file is set,
 125 the effective group ID of the new process image is set to the group ID of the new process
 126 image file. The real user ID, real group ID, and supplementary group IDs of the new
 127 process image remain the same as those of the calling process image. If B
 128 {_POSIX_SAVED_IDS} is defined, the effective user ID and effective group ID of the new B
 129 process shall be saved (as the *saved set-user-ID* and the *saved set-group-ID*) for use by
 130 the *setuid()* function.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

- 131 The new process image also inherits at least the following attributes from the calling
 132 process image:
- 133 process ID
 - 134 parent process ID
 - 135 process group ID
 - 136 terminal group ID c
 - 137 time left until an alarm clock signal (see *alarm()* §3.4.1)
 - 138 current working directory
 - 139 root directory
 - 140 file mode creation mask (see *umask()* §5.3.3)
 - 141 process signal mask (see *sigprocmask()* §3.3.5) c
 - 142 pending signals (see *sigpending()* §3.3.6) c
 - 143 *tms_utime*, *tms_stime*, *tms_cstime*, and *tms_cutime* (see *times()* §4.5.2)
- 144 Upon successful completion, the *exec* functions shall mark for update the *st_atime* field c
 145 of the file. If the *exec()* function failed but was able to locate the *process image file*, c
 146 whether the *st_atime* field is marked for update is unspecified. c
- 147 **3.1.2.3 Returns**
- 148 If one of the *exec* functions returns to the calling process image, an error has occurred;
 149 the return value shall be -1 , and *errno* shall be set to indicate the error.
- 150 **3.1.2.4 Errors**
- 151 If any of the following conditions occur, the *exec* functions shall return -1 and set *errno* B
 152 to the corresponding value: B
- 153 [E2BIG] The number of bytes used by the new process image's argument
 154 list and environment list is greater than the system-imposed limit
 155 of {ARG_MAX} bytes.
 - 156 [EACCES] Search permission is denied for a directory listed in the new
 157 process image file's path prefix, or the new process file is not a
 158 regular file, or the new process image file denies execution
 159 permission.
 - 160 [ENAMETOOLONG] c
 161 The length of the *path* or *file* argument exceeds {PATH_MAX}, or c
 162 a pathname component is longer than {NAME_MAX} while c
 163 {_POSIX_NO_TRUNC} is in effect. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

164	[ENOENT]	One or more components of the new process image file's pathname	
165		do not exist.	
166	[ENOEXEC]	The new process image file has the appropriate access permission,	
167		but is not in the proper format.	A
168			B
169	[ENOTDIR]	A component of the new process image file's path prefix is not a	
170		directory.	
171			9
172	For each of the following conditions, if the condition is detected, the functions shall		B
173	return -1 and return the corresponding value in <i>errno</i> :		B
174			B
175	[ENOMEM]	The new process image requires more memory than is allowed by	
176		the hardware or system-imposed memory management constraints.	
177	3.1.2.5 References		
178	<i>alarm()</i>	§3.4.1, <i>chmod()</i>	§5.6.4, <i>_exit()</i>
179	§3.3.1, <i>sigprocmask()</i>	§3.3.5, <i>sigpending()</i>	§3.3.6, <i>stat()</i>
180	<i>times()</i>	§4.5.2, <i>unmask()</i>	§5.3.3, Environment Description §2.7.
181	3.2 Process Termination		
182	There are three kinds of process termination:		B
183	<i>Normal termination</i>	occurs by a return from <i>main()</i> or when requested with the	B
184	<i>exit()</i> or <i>_exit()</i>	functions.	B
185	<i>Simple abnormal termination</i>	occurs when some signals are received (see	B
186	<signal.h>	§3.3.1).	B
187	<i>Abnormal termination with actions</i>	occurs when requested with the <i>abort()</i>	B
188	function or when other signals are received. Actions taken, if any, are		C
189	implementation defined.		C
190	The <i>exit()</i> and <i>abort()</i> functions shall be as described in the <i>ANSI/X3.159-198x</i>		
191	<i>Programming Language C Standard</i> (see C Language Standard §A.2.1). Both <i>exit()</i>		
192	and <i>abort()</i> shall terminate a process with the consequences specified in <i>_exit()</i>		
193	§3.2.2, except that the status made available to <i>wait()</i> or <i>wait2()</i> by <i>abort()</i> shall be that of a		
194	process terminated by the SIGABRT signal.		
195	A parent process can suspend its execution to wait for termination of a child process with		
196	the <i>wait()</i> or <i>wait2()</i> functions.		

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

197 3.2.1 Wait for Process Termination

198 Functions: *wait()*, *wait2()* 8

199 3.2.1.1 Synopsis

200 `int wait (stat_loc)`201 `int *stat_loc;`

202

203 `#include <sys/wait.h>` B204 `int wait2 (stat_loc, options)` 8205 `int *stat_loc;` 8206 `int options;` 8

207 3.2.1.2 Description

208 The header `<sys/wait.h>` defines the following arguments for the *wait2()* function: 8

<u>Constant</u>	<u>Description (<i>wait2()</i> only)</u>	
WNOHANG	return immediately if no children to wait for	8
WUNTRACED	also return status for stopped children	8

212 The *wait()* function suspends execution of a process until one of its children terminates.213 The termination of a child process causes *wait()* to return. If several child processes c214 have terminated, which child's information is returned by a call to *wait()* is unspecified. c215 Signals or implementation defined conditions may cause the return of *wait()* prior to the216 termination of a child. If a child process has terminated prior to the call on *wait()*, return

217 shall be immediate.

218 If *stat_loc* is not (int *) 0, information called *status* shall be stored in the location pointed219 to by *stat_loc* as follows:

220 If the child process terminated due to an *_exit()* function, the low order 8 bits of

221 *status* (corresponding to the octal value 0377) shall be zero, and the 8 bits

222 corresponding to the octal value 0177400 shall contain the low order 8 bits of the

223 argument that the child process passed to *_exit()* (see *_exit()* §3.2.2).

224 If the child process terminated due to a signal that was not caught, the low order 6

225 bits of *status* (corresponding to the octal value 077) shall contain the number of

226 the signal that caused the termination, and the 8 bits corresponding to the octal

227 value 0177400 shall be zero. In addition, if the bit that would be masked by the

228 octal value 0200 is set, an abnormal termination with actions occurred (see c

229 `<signal.h>` §3.3.1). c230 If the *wait()* function returned due to an implementation defined condition, the bit231 of *status* corresponding to the octal value 0100 shall be set. The value of the232 other bits of *status* are implementation defined and the child may not have233 terminated. If the child has terminated, a subsequent *wait()* function shall return

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 234 its *status*.
- 235 If a parent process terminates without waiting for its child processes to terminate, its
 236 children shall be assigned a new parent process ID corresponding to an implementation B
 237 defined system process. The *wait()* function shall only return successfully on the
 238 termination of a child process or due to an implementation defined change in status of a
 239 child process.
- 240 If the implementation supports the Job Control Option, the *wait2()* function shall be c
 241 provided as an alternate interface to provide both non-blocking status collection and the c
 242 collection of the status of children that are stopped. The *stat_loc* argument is defined as c
 243 above. If the *options* argument is zero, the behavior shall be identical to *wait()*. c
 244 Otherwise, the *options* argument consists of the logical OR of the following flags: c
- 245 WNOHANG Return immediately, even if there are no children to wait for. In c
 246 this case, a return value of zero shall indicate that no children have c
 247 terminated (or stopped, if WUNTRACED is also set). c
- 248 WUNTRACED Return the status of stopped children. If the child process has c
 249 stopped due to the delivery of a SIGTTIN, SIGTTOU, SIGTSTP, or c
 250 SIGSTOP signal, its status may be collected using this option. c
- 251 If WUNTRACED is set and the *status* of a stopped child process is reported, the 8 bits of c
 252 *status* (corresponding to the octal value 0177400) shall contain the number of the signal c
 253 that caused the process to stop and the low order 8 bits (corresponding to the octal value c
 254 0377) shall be set to the octal value 0177. c
- 255 3.2.1.3 Returns
- 256 If the *wait()* function returns due to the receipt of a signal by the calling process, a value
 257 of -1 shall be returned to the calling process and *errno* shall be set to [EINTR]. If the
 258 *wait()* function returns due to a terminated child process, the process ID of the child shall
 259 be returned to the calling process. Otherwise, a value of -1 shall be returned, and *errno*
 260 shall be set to indicate the error.
- 261 If *wait2()* is called, the WNOHANG option is used, and there are no stopped or 8
 262 terminated children, then a value of zero is returned. Otherwise, a value of -1 is returned 8
 263 and *errno* shall be set to indicate the error. 8

264 **3.2.1.4 Errors**

265 If any of the following conditions occur, the *wait()* and *wait2()* functions shall return -1 B
 266 and set *errno* to the corresponding value: B

267 [ECHILD] The calling process has no existing unwaited-for child processes.

268 [EINTR] The *wait()* function was terminated by a signal. The value pointed
 269 to by *stat_loc* may be undefined.

270 If any of the following conditions occur, the *wait2()* function shall return -1 and set B
 271 *errno* to the corresponding value: B

272 [EINVAL] The *wait2()* was called with an invalid *options* value. B

273 B

274 **3.2.1.5 References**

275 *exec* §3.1.2, *_exit()* §3.2.2, *fork()* §3.1.1, *pause()* §3.4.2, *times()* §4.5.2, *sigaction()* C
 276 §3.3.4. C

277 **3.2.2 Terminate a Process**

278 Function: *_exit()*

279 **3.2.2.1 Synopsis**

```
280 void _exit(status)
281 int status;
```

282 **3.2.2.2 Description**

283 The *_exit()* function shall terminate the calling process with the following consequences:

284 All open file descriptors in the calling process are closed.

285 If the parent process of the calling process is executing a *wait()* or *wait2()*, it is 8
 286 notified of the calling process's termination and the low order 8 bits of *status* are
 287 made available to it; see *wait* §3.2.1. C

288 If the parent process of the calling process is not executing a *wait()* or *wait2()* 8
 289 function, the exit *status* code is saved for return to the parent process whenever
 290 the parent process executes a subsequent *wait()* or *wait2()*.

291 B

292 Termination of a process does not terminate its children. Children of a terminated B
 293 process shall be assigned a new parent process ID, corresponding to an B
 294 implementation defined system process. B

295 If the implementation supports the SIGCLD signal, a SIGCLD shall be sent to the C
 296 parent process. C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 297 If the process is a controlling process, and if `{_POSIX_EXIT_SIGHUP}` is defined, c
 298 the SIGHUP signal shall be sent to each process that has a process group ID equal c
 299 to that of the calling process; otherwise, the signal shall not be sent. c
- 300 If the process is a session process group leader, and if `{_POSIX_PGID_CLEAR}` is c
 301 defined, the the process group ID shall be set to zero for each process that had a c
 302 process group ID equal to that of the calling process; otherwise, the group IDs c
 303 shall not be affected. c
- 304 If the implementation supports the Job Control Option and if the calling process 8
 305 has child processes that are stopped, they shall be sent SIGHUP and SIGCONT 8
 306 signals. 8
- 307 If the implementation supports the Job Control Option, and if the process is a c
 308 controlling process, the terminal group ID shall be cleared of all processes that c
 309 match the terminal group ID of the calling process. c
- 310 These consequences shall occur on process termination for any reason.
- 311 Application programs should use the C language function `exit()`, defined in the 9
 312 *ANSI/X3.159-198x Programming Language C Standard*, rather than `_exit()`. The 9
 313 function `_exit()` is included to clearly define the termination consequences for all 9
 314 processes. If a program reaches the end of a `main()` procedure, the return value is 9
 315 undefined. 9
- 316 **3.2.2.3 Returns**
- 317 The `_exit()` function cannot return to its caller.
- 318 **3.2.2.4 References**
- 319 `close()` §6.3.1, `sigaction()` §3.3.4, `wait` §3.2.1. c

320 3.3 Signals

321 3.3.1 Signal Names

322 3.3.1.1 Synopsis

323 `#include <signal.h>`

324 3.3.1.2 Description

325 The `<signal.h>` header declares the `sigset_t` type and the `sigaction` structure. It also
 326 defines the following symbolic constants, each of which expands to a distinct constant
 327 expression of the type `void(*)()`, whose value matches no declarable function.

<u>Symbolic Constant</u>	<u>Description</u>
<code>SIG_DFL</code>	request for default signal handling
<code>SIG_IGN</code>	request that signal be ignored

331 The type `sigset_t` is used to represent sets of signals. It is always an integral or structure
 332 type. Several functions used to manipulate objects of type `sigset_t` are defined in
 333 *sigsetops* §3.3.3.

334 The `<signal.h>` header also declares the constants that are used to refer to the signals that
 335 occur in the system. Each of the signals defined by this standard shall have distinct,
 336 positive integral values. The value zero is reserved for use as the null signal (see *kill()*
 337 §3.3.2). An implementation may define additional signals that may occur in the system.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

338 The following constants shall be defined by all implementations:

Symbolic Constant	Default Action	Required Signals	
		Description	
SIGABRT	2	abnormal termination signal, such as is initiated by the <i>abort()</i> function (as defined in the <i>ANSI/X3.159-198x Programming Language C Standard</i>)	A A A B B B B
SIGALRM	1	timeout signal, such as initiated by the <i>alarm()</i> function (see <i>alarm()</i> §3.4.1)	B B
SIGFPE	2	erroneous arithmetic operation, such as division by zero or an operation resulting in overflow	B B B
SIGHUP	1	hangup detected on controlling terminal (see <i>Modem Disconnect</i> §7.1.1.11) or death of process group leader (see <i>_exit()</i> §3.2.2)	B B B B
SIGILL	2	detection of an invalid hardware instruction	B B
SIGINT	1	interactive attention signal (see <i>Special Characters</i> §7.1.1.10)	C C
SIGKILL	1	termination signal (cannot be caught or ignored)	B B
SIGPIPE	1	write on a pipe with no readers (see <i>write()</i> §6.4.2)	C C
SIGQUIT	2	interactive termination signal (see <i>Special Characters</i> §7.1.1.10)	C C
SIGSEGV	2	detection of an invalid memory reference	B B
SIGTERM	1	termination signal	B C
SIGUSR1	1	reserved as application defined signal 1	B
SIGUSR2	1	reserved as application defined signal 2	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

367 In addition, if the implementation supports the Job Control Option, the following
 368 constants shall be defined: A
 A

Job Control Option Signals			A
Symbolic Constant	Default Action	Description	A A A
SIGCLD	3	child process terminated (see <code>_exit()</code> §3.2.2)	C C
SIGCONT	5	continue if stopped (cannot be ignored)	B
SIGSTOP	4	stop signal (cannot be caught or ignored)	B
SIGTSTP	4	interactive stop signal (see Special Characters §7.1.1.10)	C C
SIGTTIN	4	background read attempted from control terminal (see Job Access Control §7.1.1.5)	C C C
SIGTTOU	4	background write attempted to control terminal (see Job Access Control §7.1.1.5)	C C C

378 The constant SIGCLD may be defined in implementations that do not support the Job
 379 Control Option. If SIGCLD is defined, it shall behave as specified in this standard. B
 B

380 Default actions for the preceding tables are as follows: A

381 1 Simple abnormal termination (see **Process Termination** §3.2). A

382 2 Abnormal termination with actions (see **Process Termination** §3.2). A

383 3 Ignore the signal. A

384 4 Stop the process if it is currently executing; otherwise, ignore the signal. A

385 5 Continue the process if it is currently stopped; otherwise, ignore the signal. A

386 B

387 A signal is said to be generated for (or sent to) a process when the event that causes the
 388 signal first occurs. Examples of such events include detection of hardware faults, timer
 389 expiration, and terminal activity; as well as the invocation of the `kill()` function. The
 390 same event may generate signals for multiple processes. A
 A

391 Each process has an action to be taken in response to each signal defined by the system. A
 392 A signal is said to be delivered to a process when the appropriate action for the process A
 393 and signal is taken. The action taken in response to a signal is determined at the time the A
 394 signal is delivered. This determination is independent of the means by which the signal A
 395 was originally generated. A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

396 During the time between the generation of a signal and its delivery, the signal is said to
 397 be pending. Ordinarily, this interval cannot be detected by an application. However, a
 398 signal can be blocked from delivery to a process, in which case it remains pending until it
 399 is unblocked. Each process has a signal mask that defines the set of signals currently
 400 blocked from delivery to it. The signal mask for a process is initialized from that of its
 401 parent. The *sigaction()*, *sigprocmask()*, and *sigsuspend()* functions control the
 402 manipulation of the signal mask. If a subsequent occurrence of a pending signal is
 403 generated, it is implementation defined as to whether the signal is delivered more than
 404 once.

405 When SIGCONT is generated for a process, all pending stop signals (SIGSTOP, SIGTSTP,
 406 SIGTTIN, SIGTTOU) for that process shall be discarded. Conversely, when any stop
 407 signal is generated for a process, any pending SIGCONT signals for that process shall be
 408 discarded.

409 An implementation shall document any conditions not specified by this standard under
 410 which the implementation generates signals. (See Documentation §2.2.1.2.)

411 3.3.1.3 Signal Actions

412 There are three types of actions that can be associated with a signal: SIG_DFL, SIG_IGN,
 413 or a *pointer to a function*. Initially, all signals shall be set to SIG_DFL or SIG_IGN prior
 414 to entry of the *main()* routine (see *exec* §3.1.2). The actions prescribed by these values
 415 are as follows:

416 SIG_DFL — signal-specific default action

417 The default actions for the signals defined in this standard are specified in the
 418 preceding tables.

419 If the default action is to stop the process, the execution of that process is
 420 temporarily suspended. When a process stops, a SIGCLD signal shall be
 421 generated for its parent process, if the parent process has set the SA_CLDSTOP
 422 flag (see *sigaction()* §3.3.4). While a process is stopped, any additional signals
 423 that are sent to the process shall not be delivered until the process is continued.

424 An exception to this is SIGKILL, which always terminates the receiving
 425 process. Another exception is SIGCONT, which always causes the receiving
 426 process to continue. For implementations that support the Job Control Option,
 427 a process whose parent has terminated shall be sent a SIGKILL signal if the
 428 SIGTSTP, SIGTTIN, or SIGTTOU signals are generated for the process.

429 If a signal action is set to SIG_DFL while the signal is pending, the signal shall
 430 remain pending.

431 SIG_IGN — ignore signal

432 Delivery of the signal shall have no effect on the process.

433 The system shall not allow the action for the signals SIGKILL, SIGSTOP, or
 434 SIGCONT to be set to SIG_IGN.

435	If a signal action is set to SIG_IGN while the signal is pending, the pending	c
436	signal shall be discarded.	C
437	If a process sets the action for the SIGCLD signal to SIG_IGN, the behavior is	B
438	implementation defined.	B
439	<i>pointer to a function</i> — catch signal	B
440	On delivery of the signal, the receiving process is to execute the signal-	B
441	catching function at the specified address. The signal number is passed as the	B
442	first argument to the signal-catching function. Other implementation specific	B
443	and signal-specific arguments are allowed. After returning from the signal-	B
444	catching function, the receiving process shall resume execution at the point it	B
445	was interrupted.	B
446	If a signal action is set to a <i>pointer to a function</i> while the signal is pending, the	C
447	signal shall remain pending.	C
448	The action taken upon normal return from a signal-catching function for	
449	signals SIGFPE, SIGILL, or SIGSEGV is implementation defined.	B
450	The system shall not allow a process to catch the signals SIGKILL and	9
451	SIGSTOP.	9
452	If a process establishes a signal-catching function for the SIGCLD signal while	B
453	it has any child processes, the behavior is implementation defined.	B
454	If a process attempts to establish a signal-catching function for the SIGCONT	B
455	signal, the behavior is implementation defined.	B
456	When signal-catching functions are invoked asynchronously with process	B
457	execution, the behavior of some of the functions defined by this standard is	B
458	unspecified if they are called from a signal-catching function. The following	B
459	table defines a set of functions that shall be reentrant with respect to signals	B
460	(that is, applications may invoke them, without restriction, from signal-	B
461	catching functions):	B
462	<i>_exit()</i> <i>access()</i> <i>alarm()</i>	B
463	<i>chdir()</i> <i>chmod()</i> <i>chown()</i>	B
464	<i>close()</i> <i>creat()</i> <i>dup2()</i>	B
465	<i>dup()</i> <i>exec()</i> <i>fcntl()</i>	B
466	<i>fork()</i> <i>fstat()</i> <i>getegid()</i>	B
467	<i>geteuid()</i> <i>getgid()</i> <i>getgroups()</i>	B
468	<i>getpgrp()</i> <i>getpid()</i> <i>getppid()</i>	B
469	<i>getuid()</i> <i>jcgetpgrp()</i> <i>jcsetpgrp()</i>	B
470	<i>kill()</i> <i>link()</i> <i>lseek()</i>	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

471	<i>mkdir()</i>	<i>mkfifo()</i>	<i>open()</i>	B
472	<i>pause()</i>	<i>pipe()</i>	<i>read()</i>	B
473	<i>rename()</i>	<i>rmdir()</i>	<i>setgid()</i>	B
474	<i>setpgrp()</i>	<i>setuid()</i>	<i>sigaction()</i>	B
475	<i>sigaddset()</i>	<i>sigdelset()</i>	<i>sigfillset()</i>	B
476	<i>siginitset()</i>	<i>sigismember()</i>	<i>signal()</i>	B
477	<i>sigpending()</i>	<i>sigprocmask()</i>	<i>sigsuspend()</i>	B
478	<i>sleep()</i>	<i>stat()</i>	<i>tcdrain()</i>	B
479	<i>tcflow()</i>	<i>tcflush()</i>	<i>tcgetattr()</i>	B
480	<i>tcgetpgrp()</i>	<i>tcsendbreak()</i>	<i>tcsetattr()</i>	B
481	<i>tcsetpgrp()</i>	<i>time()</i>	<i>times()</i>	B
482	<i>umask()</i>	<i>uname()</i>	<i>unlink()</i>	B
483	<i>ustat()</i>	<i>utime()</i>	<i>wait2()</i>	B
484	<i>wait()</i>	<i>write()</i>		B

485 All IEEE Std 1003.1 functions not in the above table and all functions defined B
 486 in the *ANSI/X3.159-198x Programming Language C Standard* not stated to be B
 487 callable from a signal-catching function are considered to be *unsafe* with B
 488 respect to signals. If any function that is *unsafe* is interrupted by a signal- B
 489 catching function that then calls any function that is *unsafe*, the behavior is B
 490 undefined. B

491 C

492 3.3.2 Send a Signal to a Process

493 Function: *kill()*

494 3.3.2.1 Synopsis

495 `#include <signal.h>`

496 `int kill(pid, sig)`

497 `int pid, sig;`

498 3.3.2.2 Description

499 The *kill()* function shall send a signal to a process or a group of processes specified by
 500 *pid*. The signal to be sent is specified by *sig* and is either one from the list given in
 501 <signal.h> §3.3.1 or zero. If *sig* is zero (the null signal), error checking is performed but
 502 no signal is actually sent. The null signal can be used to check the validity of *pid*.

503 For a process to have permission to send a signal to a process designated by *pid*, the real 8
 504 or effective user ID of the sending process must match the real or effective user ID of the 8
 505 receiving process, unless the sending process has appropriate privileges. If both C
 506 `{_POSIX_KILL_SAVED}` and `{_POSIX_SAVED_IDS}` are defined, the saved set-user-ID C
 507 of the receiving process shall be checked in place of its effective user ID. If a receiving 9
 508 process's effective user ID has been altered through use of the `S_ISUID` mode bit (see B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

509 <sys/stat.h> §5.6.1), it may still receive a signal sent by the parent process or by a 9
510 process with the same real user ID. The calling process may be restricted from sending a c
511 signal by implementation defined constraints. c

512 If *pid* is greater than zero, *sig* shall be sent to the process whose process ID is equal to
513 *pid*.

514 If *pid* is zero, *sig* shall be sent to all processes (excluding an implementation defined set 9
515 of system processes) whose process group ID is equal to the process group ID of the c
516 sender. c

517 If *pid* is -1 , *sig* shall be sent to all processes (excluding the special set of system 8
518 processes). If `{_POSIX_KILL_PID_NEG1}` is defined, *sig* also shall be sent to the sending c
519 process; otherwise, it shall not be sent to the sending process. c

520 If *pid* is negative but not -1 , *sig* shall be sent to all processes whose process group ID is
521 equal to the absolute value of *pid*. The absolute value of *pid* shall not exceed
522 `{PID_MAX}`.

523 If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not B
524 blocked, then either *sig* or at least one pending unblocked signal shall be delivered to the B
525 sending process before the *kill()* function returns. B

526 As a single special case on implementations that support the Job Control Option, if the B
527 sending process has a controlling terminal, the *kill()* function shall allow the SIGCONT B
528 signal to be sent to any process that has the same controlling terminal as the sending B
529 process. B

530 A process may be restricted from sending a signal, including the null signal, to a c
531 particular process by implementation defined constraints. c

532 The *kill()* function is successful if the process has permission to send *sig* to any of the B
533 processes specified by *pid*. If the *kill()* function fails, no signal shall be sent. 9

534 3.3.2.3 Returns

535 Upon successful completion, the function shall return a value of zero. Otherwise, a value
536 of -1 shall be returned and *errno* shall be set to indicate the error.

537 3.3.2.4 Errors

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 538 If any of the following conditions occur, the *kill()* function shall return -1 and set *errno* B
 539 to the corresponding value: B
- 540 [EINVAL] The value of the *sig* argument is not a valid signal number.
- 541 [EPERM] The process does not have permission to send the signal to any B
 542 receiving process. B
- 543 [ESRCH] No process can be found corresponding to that specified by *pid*.
- 544 3.3.2.5 References
- 545 *getpid()* §4.1.1, *setpgroup()* §4.3.2, *sigaction()* §3.3.4, *<signal.h>* §3.3.1. B
 546
- 547 3.3.3 Manipulate Signal Sets 8
- 548 Functions: *siginitset()*, *sigfillset()*, *sigaddset()*, *sigdelset()*, *sigismember()* B
- 549 3.3.3.1 Synopsis 8
- 550 #include *<signal.h>*
- 551 int *siginitset(set)* B
 552 sigset_t **set*; B
- 553 int *sigfillset(set)* B
 554 sigset_t **set*; B
- 555 B
- 556 int *sigaddset(set, signo)* B
 557 sigset_t **set*; B
 558 int *signo*; B
- 559 int *sigdelset(set, signo)* B
 560 sigset_t **set*; B
 561 int *signo*; B
- 562 int *sigismember(set, signo)* B
 563 sigset_t **set*; B
 564 int *signo*; B
- 565 3.3.3.2 Description 8
- 566 The *sigsetops* primitives manipulate sets of signals. They operate on data objects 8
 567 addressable by the application, not on any set of signals known to the system, such as the 8
 568 set blocked from delivery to a process or the set pending for a process (see *<signal.h>* 8
 569 §3.3.1). 8
- 570 The *siginitset()* function initializes the signal set pointed to by the argument *set*, such B
 571 that all signals defined in this standard are excluded. Applications shall call *siginitset()* B
 572 at least once for each object of type *sigset_t* prior to any other use of that object. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

573 The *sigfillset()* function initializes the signal set pointed to by the argument *set*, such that B
574 all signals defined in this standard are included. B

575 The *sigaddset()* and *sigdelset()* functions respectively add and delete the individual 8
576 signal specified by the value of the argument *signo* from the signal set pointed to by the B
577 argument *set*. B

578 The *sigismember()* function tests whether the signal specified by the value of the B
579 argument *signo* is a member of the set pointed to by the argument *set*. 8

580 B

581 **3.3.3.3 Returns** 8

582 Upon successful completion, the *sigismember()* function returns a value of one if the 8
583 specified signal is a member of the specified set, or a value of zero if it is not. Upon 8
584 successful completion, the other functions return a value of zero. For all of the above B
585 functions, if an error is detected, a value of -1 is returned and *errno* is set to indicate the 8
586 error. 8

587 **3.3.3.4 Errors** 8

588 If any of the following conditions occur, the *sigaddset()*, *sigdelset()*, and *sigismember()* B
589 functions shall return -1 and set *errno* to the corresponding value: B

590 [EINVAL] The value of the *signo* argument is not a valid signal number. 8

591 B

592 **3.3.3.5 References** 8

593 *sigaction()* §3.3.4, <signal.h> §3.3.1, *sigpending()* §3.3.6, *sigprocmask()* §3.3.5, 8
594 *sigsuspend()* §3.3.7. 8

595 **3.3.4 Examine and Change Signal Action** 8

596 Function: *sigaction()* 8

597 **3.3.4.1 Synopsis** 8

598 #include <signal.h> 8

599 int sigaction (*sig*, *act*, *oact*) 8
600 int *sig*; 8
601 struct sigaction **act*, **oact*; 8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

602 3.3.4.2 Description 8

603 The *sigaction()* function allows the calling process to examine and/or specify the action 8
 604 to be taken on delivery of a specific signal. The argument *sig* specifies the signal; 8
 605 acceptable values are defined in <signal.h> §3.3.1. 8

606 The structure *sigaction*, used to describe an action to be taken, is defined in the header 8
 607 <signal.h> to include at least the following members: 8

Member Type	Member Name	Description	
<i>void (*)()</i>	<i>sa_handler</i>	SIG_DFL, SIG_IGN, or pointer to a function	8
<i>sigset_t</i>	<i>sa_mask</i>	set of signals to be blocked during execution of signal-catching function	8
<i>int</i>	<i>sa_flags</i>	special flags to be used when delivering signal	8

614 If the argument *act* is not NULL, it points to a structure specifying the action to be taken 8
 615 when delivering the specified signal. If the argument *oact* is not NULL, the action 8
 616 previously associated with the signal is stored in the location pointed to by the argument 8
 617 *oact*. If the argument *act* is NULL, signal handling is unchanged; thus, the call can be 8
 618 used to inquire about the current handling of a given signal. 8

619 The *sa_flags* field can be used to modify the delivery of the specified signal. If *sig* is 8
 620 SIGCLD and the implementation supports the Job Control Option, the following flag bit, 8
 621 defined in the header <signal.h>, can be set in *sa_flags*: 8

Symbolic Constant	Description	
SA_CLDSTOP	Also generate SIGCLD when children stop	8

625 An implementation may define additional flag bits in the *sa_flags* field. C
 626 9

627 When a signal is caught by a signal-catching function installed by the *sigaction()* 8
 628 function, a new signal mask is calculated and installed for the duration of the signal- 8
 629 catching function (or until a *sigprocmask()* or *sigsuspend()* function is made). This 8
 630 mask is formed by taking the union of the current signal mask and the set associated with 8
 631 the action for the signal being delivered, and then including the signal being delivered. If 8
 632 and when the user's signal handler returns normally, the original signal mask is restored. 8

633 Once an action is installed for a specific signal, it remains installed until another action is 8
 634 explicitly requested (by another call to the *sigaction()* function), or until one of the *exec* 8
 635 functions is called. 8

636 B

637	The set of signals specified by the <i>sa_mask</i> field pointed to by the argument <i>act</i> is not	8
638	allowed to block those signals that cannot be ignored, as defined in <signal.h> §3.3.1.	C
639	This shall be enforced by the system without causing an error to be indicated.	C
640	If the <i>sigaction()</i> function fails, no new signal handler is installed.	9
641	3.3.4.3 Returns	8
642	Upon successful completion a value of zero is returned. Otherwise, a value of -1 is	8
643	returned and <i>errno</i> is set to indicate the error.	8
644	3.3.4.4 Errors	8
645	If any of the following conditions occur, the <i>sigaction()</i> function shall return -1 and set	B
646	<i>errno</i> to the corresponding value:	B
647	[EINVAL] The value of the <i>sig</i> argument is not a valid signal number, or an	8
648	attempt is made to supply an action for a signal that cannot be	8
649	caught or ignored. See <signal.h> §3.3.1.	8
650		B
651	3.3.4.5 References	8
652	<i>kill()</i> §3.3.2, <signal.h> §3.3.1, <i>sigprocmask()</i> §3.3.5, <i>sigsetops</i> §3.3.3, <i>sigsuspend()</i>	8
653	§3.3.7.	8
654	3.3.5 Examine and Change Blocked Signals	8
655	Function: <i>sigprocmask()</i>	8
656	3.3.5.1 Synopsis	8
657	#include <signal.h>	8
658	int sigprocmask (<i>how</i> , <i>set</i> , <i>oset</i>)	8
659	int <i>how</i> ;	8
660	sigset_t * <i>set</i> , * <i>oset</i> ;	8
661	3.3.5.2 Description	8
662	The <i>sigprocmask()</i> function is used to examine and/or change the calling process's signal	8
663	mask. If the value of the argument <i>set</i> is not NULL, it points to a set of signals to be used	8
664	to change the currently blocked set.	8
665	The value of the argument <i>how</i> indicates the manner in which the set is changed, and	B
666	shall consist of one of the following values, as defined in the header <signal.h> §3.3.1:	B
667	SIG_BLOCK The resulting set shall be the union of the current set and the	B
668	signal set pointed to by the argument <i>set</i> .	B
669	SIG_UNBLOCK The resulting set shall be the intersection of the current set and	B
670	the complement of the signal set pointed to by the argument	B
671	<i>set</i> .	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

672 SIG_SETMASK The resulting set shall be the signal set pointed to by the B
673 argument *set*. B

674 If the argument *oset* is not NULL, the previous mask is stored in the space pointed to by 8
675 *oset*. If the value of the argument *set* is NULL, the value of the argument *how* is not 8
676 significant and the process's signal mask is unchanged; thus, the call can be used to 8
677 enquire about currently blocked signals. 8

678 If there are any pending unblocked signals after the call to the *sigprocmask()* function, at B
679 least one of those signals shall be delivered before the *sigprocmask()* function returns. B

680 It is not possible to block those signals that cannot be ignored, as documented in 8
681 <signal.h> §3.3.1; this shall be enforced by the system without causing an error to be C
682 indicated. C

683 If the *sigprocmask()* function fails, the process's signal mask is not changed. 9

684 3.3.5.3 Returns 8
685 Upon successful completion a value of zero is returned. Otherwise, a value of -1 is 8
686 returned and *errno* is set to indicate the error. 8

687 3.3.5.4 Errors 8
688 If any of the following conditions occur, the *sigprocmask()* function shall return -1 and B
689 set *errno* to the corresponding value: B

690 [EINVAL] The value of the *how* argument is not equal to one of the defined 8
691 values. 8

692 B

693 3.3.5.5 References 8
694 *sigaction()* §3.3.4, <signal.h> §3.3.1, *sigpending()* §3.3.6, *sigsetops* §3.3.3, 8
695 *sigsuspend()* §3.3.7. 8

696 3.3.6 Examine Pending Signals 8
697 Function: *sigpending()* 8

698 3.3.6.1 Synopsis 8
699 #include <signal.h> 8
700 int sigpending(*set*) 8
701 sigset_t **set*; 8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

702	3.3.6.2 Description	8
703	The <i>sigpending()</i> function shall store the set of signals that are blocked from delivery	8
704	and pending for the calling process, in the space pointed to by the argument <i>set</i> .	8
705	3.3.6.3 Returns	8
706	Upon successful completion a value of zero is returned. Otherwise, a value of -1 is	8
707	returned and <i>errno</i> is set to indicate the error.	8
708	3.3.6.4 Errors	C
709	This standard does not specify any error conditions that are required to be detected for	C
710	the <i>sigpending()</i> function. Some errors may be detected under implementation defined	C
711	conditions.	C
712	3.3.6.5 References	8
713	<signal.h> §3.3.1, <i>sigprocmask()</i> §3.3.5, <i>sigsetops</i> §3.3.3.	8
714	3.3.7 Wait for a Signal	8
715	Function: <i>sigsuspend()</i>	8
716	3.3.7.1 Synopsis	8
717	#include <signal.h>	8
718	int sigsuspend (<i>sigmask</i>)	8
719	sigset_t * <i>sigmask</i> ;	8
720	3.3.7.2 Description	8
721	The <i>sigsuspend()</i> function replaces the process's signal mask with the set of signals	8
722	pointed to by the argument <i>sigmask</i> and then suspends the process until delivery of a	B
723	signal whose action is either to execute a signal-catching function or to terminate the	B
724	process.	B
725	If the action is to terminate the process, the <i>sigsuspend()</i> function shall not return. If the	B
726	action is to execute a signal-catching function, the <i>sigsuspend()</i> shall return after the	B
727	signal-catching function returns, with the signal mask restored to the set that existed prior	B
728	to the <i>sigsuspend()</i> call.	B
729		B
730	It is not possible to block those signals that cannot be ignored, as documented in	8
731	<signal.h> §3.3.1; this shall be enforced by the system without causing an error to be	C
732	indicated.	C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

733	3.3.7.3 Returns		
734	Since the <i>sigsuspend()</i> function suspends process execution indefinitely, there is no		9
735	successful completion return value. A value of <i>-1</i> is returned and <i>errno</i> is set to indicate		B
736	the error.		B
737	3.3.7.4 Errors		9
738	If any of the following conditions occur, the <i>sigsuspend()</i> function shall return <i>-1</i> and set		B
739	<i>errno</i> to the corresponding value:		B
740	[EINTR] A signal is caught by the calling process and control is returned		8
741	from the signal-catching function.		8
742			B
743	3.3.7.5 References		8
744	<i>pause()</i> §3.4.2, <i>sigaction()</i> §3.3.4, <i><signal.h></i> §3.3.1, <i>sigpending()</i> §3.3.6,		8
745	<i>sigprocmask()</i> §3.3.5, <i>sigsetops</i> §3.3.3.		8
746	3.4 Timer Operations		
747	A process can suspend itself for a specific period of time with the <i>sleep()</i> function or		
748	suspend itself indefinitely with the <i>pause()</i> function until a signal arrives. The <i>alarm()</i>		
749	function schedules a signal to arrive at a specific time, so a <i>pause()</i> suspension need not		
750	be indefinite.		
751	3.4.1 Process Alarm Clock		
752	Function: <i>alarm()</i>		
753	3.4.1.1 Synopsis		
754	unsigned int <i>alarm(seconds)</i>		C
755	unsigned int <i>seconds</i> ;		C
756	3.4.1.2 Description		
757	The <i>alarm()</i> function shall instruct the calling process's alarm clock to send the signal		
758	SIGALRM to the calling process after the number of real time seconds specified by		C
759	<i>seconds</i> have elapsed; see <i>signal()</i> .		
760	Processor scheduling delays may cause the process to not actually begin handling the		9
761	signal until after the desired time. Also, an alarm may occur up to one second early.		9
762	Alarm requests are not stacked; successive calls reset the calling process's alarm clock.		
763	If <i>seconds</i> is 0, any previously made <i>alarm()</i> request is canceled.		C

764	3.4.1.3 Returns	
765	The <i>alarm()</i> function shall return the amount of time remaining in the calling process's	B
766	alarm clock from the previous <i>alarm()</i> request or zero if there is no previous <i>alarm()</i>	
767	request.	
768	3.4.1.4 References	
769	<i>exec</i> §3.1.2, <i>fork()</i> §3.1.1, <i>pause()</i> §3.4.2, <i>sigaction()</i> §3.3.4.	C
770	3.4.2 Suspend Process Execution	
771	Function: <i>pause()</i>	
772	3.4.2.1 Synopsis	
773	int pause ()	
774	3.4.2.2 Description	
775	The <i>pause()</i> function suspends the calling process until delivery of a signal whose action	B
776	is either to execute a signal-catching function or to terminate the process.	B
777	If the action is to terminate the process, the <i>pause()</i> function shall not return.	B
778	If the action is to execute a signal-catching function, the <i>pause()</i> function shall return	B
779	after the signal-catching function returns.	B
780	3.4.2.3 Returns	
781	Since the <i>pause()</i> function suspends process execution indefinitely, there is no successful	9
782	completion return value. A value of -1 is returned and <i>errno</i> is set to indicate the error.	9
783	3.4.2.4 Errors	9
784	If any of the following conditions occur, the <i>pause()</i> function shall return -1 and set	B
785	<i>errno</i> to the corresponding value:	B
786	[EINTR] A signal is caught by the calling process and control is returned	9
787	from the signal-catching function.	9
788	3.4.2.5 References	
789	<i>alarm()</i> §3.4.1, <i>kill()</i> §3.3.2, <i>sigaction()</i> §3.3.4, <i>wait</i> §3.2.1.	C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

790 3.4.3 Delay Process Execution

791 Function: *sleep()*

792 3.4.3.1 Synopsis

793 unsigned int *sleep (seconds)*794 unsigned int *seconds*;

795 3.4.3.2 Description

796 The *sleep()* function shall cause the current process to be suspended from execution for
 797 the number of *seconds* specified by the argument. The actual suspension time may be
 798 less than that requested for two reasons:

- 799 1. because of timer imprecision, and 8
- 800 2. because any caught signal shall terminate the *sleep()* function following execution
 801 of that signal's catching routine.

802 The suspension time may be longer than requested by an arbitrary amount due to the
 803 scheduling of other activity in the system.

804 The routine shall behave as if implemented by setting an alarm signal and pausing until it
 805 (or some other signal) occurs. The previous state of the alarm signal shall be saved and
 806 restored. The calling process may have set up an alarm signal before calling *sleep()*; if c
 807 the *sleep()* time exceeds the time until such alarm signal, the process sleeps only until
 808 the alarm signal would have occurred, and the caller's alarm catch routine is executed
 809 just before the *sleep()* routine returns, but if the *sleep()* time is less than the time until
 810 such alarm, the prior alarm time shall go off at the same time it would have without the
 811 intervening *sleep()*.

812 3.4.3.3 Returns

813 The value returned by the *sleep()* function shall be the unslept amount (the requested
 814 time minus the time actually slept). This return value may be non-zero in cases where c
 815 the caller had an alarm set to go off earlier than the end of the requested time, or where
 816 *sleep()* was interrupted due to another caught signal.

817 3.4.3.4 References

818 *alarm()* §3.4.1, *pause()* §3.4.2, *sigaction()* §3.3.4. c

4. Process Environment

1 4.1 Process Identification

2 4.1.1 Get Process and Parent Process IDs

3 Functions: *getpid()*, *getppid()*

4 4.1.1.1 Synopsis

5 `int getpid ()`

6 `int getppid ()`

7 4.1.1.2 Description

8 The *getpid()* function returns the process ID of the calling process.

9 The *getppid()* function returns the parent process ID of the calling process.

10 4.1.1.3 References

11 *exec* §3.1.2, *fork()* §3.1.1, *kill()* §3.3.2.

8

12 4.2 User Identification

13 4.2.1 Get Real User, Effective User, Real Group, and Effective Group IDs

14 Functions: *getuid()*, *geteuid()*, *getgid()*, *getegid()*

15 4.2.1.1 Synopsis

16 `#include <sys/types.h>`

B

17 `uid_t getuid ()`

8

18 `uid_t geteuid ()`

8

19 `uid_t getgid ()`

8

20 `uid_t getegid ()`

8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

21 4.2.1.2 Description

22 The *getuid()* function returns the real user ID of the calling process.23 The *geteuid()* function returns the effective user ID of the calling process.24 The *getgid()* function returns the real group ID of the calling process.25 The *getegid()* function returns the effective group ID of the calling process.

26 4.2.1.3 References

27 *setuid()* §4.2.2.

28 4.2.2 Set User and Group IDs

29 Functions: *setuid()*, *setgid()*

30 4.2.2.1 Synopsis

31 `#include <sys/types.h>` B32 `int setuid (uid)` C33 `uid_t uid;` 834 `int setgid (gid)` C35 `uid_t gid;` 8

36 4.2.2.2 Description

37 If `{_POSIX_SAVED_IDS}` is defined: C38 If the process has appropriate privileges, the *setuid()* function sets the real user C
39 ID, effective user ID, and the saved set-user-ID to *uid*. C40 If the process does not have appropriate privileges, but *uid* is equal to the real C
41 user ID or the saved set-user-ID, the *setuid()* function sets the effective user ID to C
42 *uid*; the real user ID and saved set-user-ID remain unchanged. C43 If the process has appropriate privileges, the *setgid()* function sets the real group C
44 ID, effective group ID, and the saved set-group-ID to *gid*. C45 If the process does not have appropriate privileges, but *gid* is equal to the real C
46 group ID or the saved set-group-ID, the *setgid()* function sets the effective group C
47 ID to *gid*; the real group ID and saved set-group-ID remain unchanged. C

48 Otherwise: C

49 If the process has appropriate privileges, the *setuid()* function sets the real user ID C
50 and effective user ID to *uid*. C51 If the process does not have appropriate privileges, but *uid* is equal to the real C
52 user ID, the *setuid()* function sets the effective user ID to *uid*; the real user ID C
53 remains unchanged. C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

54	If the process has appropriate privileges, the <i>setgid()</i> function sets the real group	c
55	ID and effective group ID to <i>gid</i> .	c
56	If the process does not have appropriate privileges, but <i>gid</i> is equal to the real	c
57	group ID, the <i>setgid()</i> function sets the effective group ID to <i>gid</i> ; the real group	c
58	ID remains unchanged.	c
59	4.2.2.3 Returns	
60	Upon successful completion, a value of zero is returned. Otherwise, a value of -1 is	
61	returned and <i>errno</i> is set to indicate the error.	
62	4.2.2.4 Errors	
63	If any of the following conditions occur, the <i>setuid()</i> function shall return -1 and set	B
64	<i>errno</i> to the corresponding value:	B
65	[EINVAL] The value of the <i>uid</i> argument is less than zero or exceeds	
66	{UID_MAX}.	
67	[EPERM] The process does not have appropriate privileges and <i>uid</i> does not	B
68	match the real user ID or, if {_POSIX_SAVED_IDS} is defined, the	c
69	saved set-user-ID.	c
70	If any of the following conditions occur, the <i>setgid()</i> function shall return -1 and set	B
71	<i>errno</i> to the corresponding value:	B
72	[EINVAL] The value of the <i>gid</i> argument is less than zero or exceeds	
73	{UID_MAX}.	
74	[EPERM] The process does not have appropriate privileges and <i>gid</i> does not	B
75	match the real group ID or, if {_POSIX_SAVED_IDS} is defined,	c
76	the saved set-group-ID.	c
77		B
78	4.2.2.5 References	
79	<i>exec</i> §3.1.2, <i>getuid()</i> §4.2.1.	
80	4.2.3 Get Supplementary Group IDs	
81	Function: <i>getgroups()</i>	
82	4.2.3.1 Synopsis	
83	<code>#include <sys/types.h></code>	B
84	<code>int getgroups (gidsetsize, grouplist)</code>	
85	<code>int gidsetsize;</code>	
86	<code>uid_t grouplist[];</code>	c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

87 4.2.3.2 Description

88 The *getgroups()* function fills in the array *grouplist* with the supplementary group IDs of A
 89 the calling process. The *gidsetsize* argument gives the length of the supplied array
 90 *grouplist*. The actual number of supplementary group IDs is returned. The values of
 91 array entries with indices larger than or equal to the returned value are undefined. As a c
 92 special case, if the *gidsetsize* argument is zero, *getgroups()* returns the number of c
 93 supplementary group IDs associated with the calling process without modifying the array c
 94 pointed to by the *grouplist* argument. c

95 Implementation of *getgroups()* is optional on systems that have defined
 96 {NGROUPS_MAX} to be zero.

97 4.2.3.3 Returns

98 The number of supplementary group IDs is returned if successful. A return value of -1
 99 indicates failure and *errno* is set to indicate the error.

100 4.2.3.4 Errors

101 If any of the following conditions occur, the *getgroups()* function shall return -1 and set B
 102 *errno* to the corresponding value: B

103 [EINVAL] The *gidsetsize* argument is less than the number of supplementary
 104 group IDs.

105 B

106 4.2.3.5 References

107 *setgid()* §4.2.2.

108 4.2.4 Get User Name

109 Functions: *getlogin()*, *cuserid()*

110 4.2.4.1 Synopsis

111 char *getlogin ()

112 #include <stdio.h>

113 char *cuserid (s)

114 char *s;

115 B

116 4.2.4.2 Description

117 These functions return a string giving a name of the user associated with the current
118 process. The *cuserid()* function returns a name associated with the effective user ID of
119 the process, and the *getlogin()* function returns the name associated by the login activity
120 with the control terminal. c

121 The recommended procedure is either to call the *cuserid()* function, or to call *getlogin()*
122 and, if it fails, to call the *getpwuid()* function with the value returned by the *getuid()*
123 function. 8

124 The *getlogin()* function returns a pointer to the user's login name. The same user ID may
125 be shared by several login names. Therefore, to ensure that the correct password
126 database entry is found, the *getlogin()* function should be used with the *getpwnam()*
127 function.

128 If *getlogin()* returns a non-NULL pointer, then that pointer is to the name the user logged
129 in under, even if there are several login names with the same user ID.

130 The *cuserid()* function generates a character representation of the login name of the
131 owner of the current process. If *s* is not a NULL pointer, it is assumed that *s* points to an
132 array of at least `L_cuserid` characters; the representation is returned in this array. The
133 symbolic constant `L_cuserid` is defined in `<stdio.h>`, and shall have a value greater than
134 zero. c
c

135 4.2.4.3 Returns

136 The *getlogin()* function returns a pointer to a string containing the user's login name, or a
137 NULL pointer if the user's login name cannot be found.

138 If *s* is a NULL pointer, the result from *cuserid()* is generated in an area that may be
139 static, the address of which is returned. If the login name cannot be found, *cuserid()*
140 returns NULL. If *s* is not a NULL pointer, *s* is returned. If the login name cannot be
141 found, the null character '\0' shall be placed at **s*.

142 The return value from *getlogin()* may point to static data that is overwritten by each call.

143 The implementation of the *cuserid()* function may use the *getpwnam()* function, so the
144 results of a user's call to either routine may be overwritten by a subsequent call to the
145 other routine.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 146 **4.2.4.4 Errors** c
 147 This standard does not specify any error conditions that are required to be detected for c
 148 the *cuserid()* function. Some errors may be detected under implementation defined c
 149 conditions. c
- 150 **4.2.4.5 References**
 151 *getpwent()* §9.2.2, *getpwuid()* §9.2.2.
- 152 **4.3 Process Groups**
- 153 **4.3.1 Get Process Group ID**
 154 Function: *getpgrp()*
- 155 **4.3.1.1 Synopsis**
 156 `int getpgrp ()`
- 157 **4.3.1.2 Description**
 158 The *getpgrp()* function returns the process group ID of the calling process.
- 159 **4.3.1.3 References**
 160 *setpgrp()* §4.3.2, *sigaction()* §3.3.4. c
- 161 **4.3.2 Set Process Group ID**
 162 Function: *setpgrp()*
- 163 **4.3.2.1 Synopsis**
 164 `int setpgrp ()`
- 165 **4.3.2.2 Description**
 166 The *setpgrp()* function shall set the process group ID of the calling process to the process c
 167 ID of the calling process and return the new process group ID. If the calling process is c
 168 not already the process group leader, it becomes a session process group leader and c
 169 releases its controlling terminal by clearing the terminal group ID. c
- 170 **4.3.2.3 Returns**
 171 The *setpgrp()* function returns the value of the new process group ID.
- 172 **4.3.2.4 References**
 173 *exec* §3.1.2, *_exit()* §3.2.2, *fork()* §3.1.1, *getpid()* §4.1.1, *kill()* §3.3.2, *sigaction()* c
 174 §3.3.4. c

175	4.3.3 Set Process Group ID for Job Control		A
176	Function: <i>jcsetpgrp()</i>		A
177	4.3.3.1 Synopsis		A
178	int jcsetpgrp (<i>pgrp</i>)		A
179	int <i>pgrp</i>;		A
180	4.3.3.2 Description		A
181	This function is provided if the implementation supports the Job Control Option.		A
182	The <i>jcsetpgrp()</i> function shall set the process group ID of the calling process to <i>pgrp</i> . If	B	
183	<i>pgrp</i> is equal to the process ID of the calling process, the calling process becomes a job	B	
184	control process group leader unless the process is already the process group leader.	C	
185	4.3.3.3 Returns		A
186	Upon successful completion, the <i>jcsetpgrp()</i> function returns a value of zero. Otherwise,	A	
187	a value of -1 is returned and <i>errno</i> is set to indicate the error.	A	
188	4.3.3.4 Errors		A
189	If any of the following conditions occur, the <i>jcsetpgrp()</i> function shall return -1 and set	B	
190	<i>errno</i> to the corresponding value:	B	
191	[EINVAL] The value of the <i>pgrp</i> argument is less than or equal to zero or	A	
192	exceeds {PID_MAX}.	A	
193	The calling process is the process group leader and the <i>pgrp</i>	C	
194	argument does not match the process ID.	C	
195	[EPERM] The value of the <i>pgrp</i> argument is greater than zero and less than	A	
196	or equal to {PID_MAX} and there are processes already in the	C	
197	process group indicated by <i>pgrp</i> and none of these processes have	B	
198	the same controlling terminal as the calling process.	B	
199	[ENOTTY] The calling process does not have a controlling terminal.	B	
200	4.3.3.5 References		A
201	<i>tcsetpgrp()</i> §7.2.4.		A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

202 4.4 System Identification

203 4.4.1 System Name

204 Function: *uname()*

205 4.4.1.1 Synopsis

206 `#include <sys/utsname.h>`207 `int uname (name)`208 `struct utsname *name;`

209 4.4.1.2 Description

210 The *uname()* function stores information identifying the current operating system in the
211 structure pointed to by the argument *name*.212 The structure *utsname* is defined in the header <sys/utsname.h>, and contains at least the 8
213 following members: 8

Member Name	Description	8
<i>sysname</i>	Name of this implementation of the operating system	8
<i>nodename</i>	Name of this node within an implementation specified communications network	8
<i>release</i>	Current release level of this implementation	8
<i>version</i>	Current version level of this release	8
<i>machine</i>	Name of the hardware type that the system is running on	8

226 Each of these data items is a null-terminated character array. Additional, implementation 8
227 defined, information may also be included in the structure. 8228 The format of each member is implementation defined. The system documentation (see B
229 Documentation §2.2.1.2) shall specify the source and format of each member and may
230 specify the range of values for each member.

231 **4.4.1.3 Returns**

232 Upon successful completion, a non-negative value is returned. Otherwise, a value of -1
233 is returned and *errno* is set to indicate the error.

234

B

235 **4.4.1.4 Errors**

236 This standard does not specify any error conditions that are required to be detected for
237 the *uname()* function. Some errors may be detected under implementation defined
238 conditions.

C

C

C

C

239 **4.5 Time**240 **4.5.1 Get System Time**241 Function: *time()*242 **4.5.1.1 Synopsis**243 `#include <time.h>`

C

244

B

245 `time_t time (tloc)`246 `time_t *tloc;`247 **4.5.1.2 Description**

248 The *time()* function returns the value of time in seconds since the Epoch (see Epoch
249 §2.3). C

250 If the argument *tloc* is not a NULL pointer, the return value is also stored in the location
251 pointed to by *tloc*. B

252 **4.5.1.3 Returns**

253 Upon successful completion, *time()* returns the value of time. Otherwise, a value of
254 $((\text{time_t})-1)$ is returned and *errno* is set to indicate the error.

255

B

256 **4.5.1.4 Errors**

257 This standard does not specify any error conditions that are required to be detected for
258 the *time()* function. Some errors may be detected under implementation defined
259 conditions. C

C

C

C

C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

260 4.5.2 Process Times

261 Function: *times()*

262 4.5.2.1 Synopsis

263 `#include <sys/types.h>`264 `#include <sys/times.h>`265 `clock_t times (buffer)`266 `struct tms *buffer;`

8

267 4.5.2.2 Description

268 The *times()* function shall fill the structure pointed to by *buffer* with time-accounting
 269 information. The *tms* structure is defined in `<sys/times.h>`; it shall contain at least the
 270 following members:

Member Type	Member Name	Description	
<i>clock_t</i>	<i>tms_utime</i>	User CPU time	8
<i>clock_t</i>	<i>tms_stime</i>	System CPU time	8
<i>clock_t</i>	<i>tms_cutime</i>	User CPU time of descendants	8
<i>clock_t</i>	<i>tms_cstime</i>	System CPU time of descendants	8

277 All times are in {CLK_TCK}ths of a second. Additional data elements may also be
 278 declared in this structure. c

279 The times of a child process are included in the times of the parent when a *wait()* or
 280 *wait2()* function returns the process ID of a terminated child. See *wait* §3.2.1. If a child
 281 process has not waited for its terminated children, their times shall not be included in its
 282 times. 8

283 The value *tms_utime* is the CPU time used while executing instructions of the calling
 284 process. c

285 The value *tms_stime* is the CPU time used by the system on behalf of the calling process.

286 The value *tms_cutime* is the sum of the *tms_utimes* and *tms_cutimes* of the child
 287 processes.

288 The value *tms_cstime* is the sum of the *tms_stimes* and *tms_cstimes* of the child
 289 processes.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

290 **4.5.2.3 Returns**

291 Upon successful completion, *times()* shall return the elapsed real time, in {CLK_TCK}ths
 292 of a second, since an arbitrary point in the past (for example, system start-up time). This
 293 point does not change from one invocation of *times()* within the process to another. The
 294 return value may overflow the possible range of type *clock_t*. If the *times()* function
 295 fails, a value of ((*clock_t*) - 1) is returned and *errno* is set to indicate the error. 8

296

B

297 **4.5.2.4 References**

298 *exec* §3.1.2, *fork()* §3.1.1, *time()* §4.5.1, *wait()* §3.2.1.

299 **4.6 Environment Variables**300 **4.6.1 Environment Access**

301 Function: *getenv()*

302 **4.6.1.1 Synopsis**

```
303         char *getenv (name)
304         char *name;
```

305 **4.6.1.2 Description**

306 The *getenv()* function searches the environment list (see **Environment Description**
 307 §2.7) for a string of the form *name=value* and returns a pointer to *value* if such a string is
 308 present. If the specified *name* cannot be found, a NULL pointer is returned.

309 **4.6.1.3 Errors**

310 This standard does not specify any error conditions that are required to be detected for
 311 the *getenv()* function. Some errors may be detected under implementation defined
 312 conditions. c

313 **4.6.1.4 References**

314 *environ* §3.1.2, **Environment Description** §2.7.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

315 4.7 Terminal Identification

316 4.7.1 Generate Terminal Pathname

317 Function: *ctermid()*

318 4.7.1.1 Synopsis

319 `#include <stdio.h>`320 `char *ctermid (s)`321 `char *s;`

322

B

323 4.7.1.2 Description

324 The *ctermid()* function generates a string that, when used as a pathname, refers to the
325 controlling terminal for the current process.326 If the *ctermid()* function returns a pathname, access to the file is not guaranteed. 9

327 4.7.1.3 Returns

328 If *s* is a NULL pointer, the string is stored in an internal static area, the contents of which
329 may be overwritten at the next call to *ctermid()*, and the address of which is returned;
330 otherwise *s* is assumed to point to a character array of at least `L_ctermid` elements; the
331 string is placed in this array and the value of *s* is returned. The symbolic constant
332 `L_ctermid` is defined in `<stdio.h>`, and shall have a value greater than zero. C333 The *ctermid()* function shall return an empty string if the pathname for the controlling C
334 terminal cannot be determined. C

335

9

336 4.7.1.4 Errors C

337 This standard does not specify any error conditions that are required to be detected for C
338 the *ctermid()* function. Some errors may be detected under implementation defined C
339 conditions. C

340 4.7.1.5 References

341 *ttyname()* §4.7.2.

342 **4.7.2 Determine Terminal Device Name**343 Functions: *ttyname()*, *isatty()*344 **4.7.2.1 Synopsis**345 `char *ttyname (fildes)`346 `int fildes;`347 `int isatty (fildes)`348 `int fildes;`349 **4.7.2.2 Description**350 The *ttyname()* function returns a pointer to a string containing a null-terminated c
351 pathname of the terminal associated with file descriptor *fildes*. 8352 The return value of *ttyname()* may point to static data that is overwritten by each call. 9353 The *isatty()* function returns 1 if *fildes* is a valid file descriptor associated with a 8
354 terminal, zero otherwise. 8355 **4.7.2.3 Returns**356 The *ttyname()* function returns a NULL pointer if *fildes* is not a valid file descriptor
357 associated with a terminal device.

358 9

359 **4.7.2.4 Errors** C360 This standard does not specify any error conditions that are required to be detected for C
361 the *ttyname()* function. Some errors may be detected under implementation defined C
362 conditions. C363 **4.8 Configurable System Variables** B364 **4.8.1 Get Configurable System Variables** B365 Function: *sysconf()* B366 **4.8.1.1 Synopsis** B367 `#include <unistd.h>` B368 `long sysconf (name)` C369 `int name;` B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

370 4.8.1.2 Description B

371 The *sysconf()* function provides a method for the application to determine the current B
 372 value of a configurable system limit or option (*variable*). B

373 The *name* argument represents the system variable to be queried. The following table B
 374 lists the system variables from `<limits.h>` §2.9 or `<unistd.h>` §2.10 that can be returned B
 375 by *sysconf()*, and the symbolic constants, defined in `<unistd.h>`, that are the B
 376 corresponding values used for *name*: B

Variable	<i>name</i> Value	B
ARG_MAX	_SC_ARG_MAX	B
CHILD_MAX	_SC_CHILD_MAX	B
CLK_TCK	_SC_CLK_TCK	C
NGROUPS_MAX	_SC_NGROUPS_MAX	C
OPEN_MAX	_SC_OPEN_MAX	C
PASS_MAX	_SC_PASS_MAX	C
PID_MAX	_SC_PID_MAX	C
UID_MAX	_SC_UID_MAX	C
_POSIX_EXIT_SIGHUP	_SC_EXIT_SIGHUP	C
_POSIX_JOB_CONTROL	_SC_JOB_CONTROL	C
_POSIX_KILL_PID_NEG1	_SC_KILL_PID_NEG1	C
_POSIX_KILL_SAVED	_SC_KILL_SAVED	C
_POSIX_PGID_CLEAR	_SC_PGID_CLEAR	C
_POSIX_SAVED_IDS	_SC_SAVED_IDS	C
_POSIX_VERSION	_SC_VERSION	C

406 4.8.1.3 Returns B

407 If the variable corresponding to *name* is not defined on the system, or if *name* is an B
 408 invalid value, the *sysconf()* function returns `-1`. B

409 Otherwise, the *sysconf()* function returns the current variable value on the system. The B
 410 value returned shall not be more restrictive than the corresponding value described to the B
 411 application when it was compiled with the implementation's `<limits.h>` §2.9 or B
 412 `<unistd.h>` §2.10. The value shall not change during the lifetime of the calling process. B

5. Files and Directories

1 The functions in this section perform the operating system services dealing with the c
2 creation and removal of files and directories and the detection and modification of their c
3 characteristics. They also provide the primary methods a process will use to gain access c
4 to files and directories for subsequent I/O operations (see Input and Output Primitives c
5 §6). c

6 5.1 Directories

7 5.1.1 Format of Directory Entries

8 5.1.1.1 Synopsis

9 #include <sys/types.h> B
10 #include <dirent.h> B

11 5.1.1.2 Description

12 The header <dirent.h> defines a structure and a defined type used by the *directory*
13 routines.

14 A
15 The internal format of directories is implementation defined.

16 The routine *readdir()* returns a pointer to an object of type *struct dirent* that includes the B
17 member: B

<u>Member</u>	<u>Member</u>	<u>Description</u>	
<u>Type</u>	<u>Name</u>		
char []	d_name	Null-terminated filename	B B B B

22 The character array *d_name* is of unspecified size, but the number of characters B
23 preceding the terminating null character shall not exceed {NAME_MAX}. B

24 Additional, implementation defined, structure elements may also be declared in this
25 structure by the header <dirent.h>. C

26 C

27		9
28	5.1.1.3 References	
29	<i>directory</i> §5.1.2.	A
30	5.1.2 Directory Operations	
31	Functions: <i>opendir()</i> , <i>readdir()</i> , <i>rewinddir()</i> , <i>closedir()</i>	
32	5.1.2.1 Synopsis	
33	#include <sys/types.h>	B
34	#include <dirent.h>	B
35	DIR *opendir (<i>dirname</i>)	B
36	char * <i>dirname</i> ;	B
37	struct dirent *readdir (<i>dirp</i>)	B
38	DIR * <i>dirp</i> ;	B
39	void rewinddir (<i>dirp</i>)	B
40	DIR * <i>dirp</i> ;	B
41	int closedir (<i>dirp</i>)	B
42	DIR * <i>dirp</i> ;	B
43	5.1.2.2 Description	
44	The type DIR, which is defined in the header <dirent.h> §5.1.1, represents a <i>directory</i>	B
45	<i>stream</i> , which is an ordered sequence of all the directory entries in a particular directory.	C
46	Directory entries represent files; files may be removed from a directory or added to a	C
47	directory asynchronous to the operations described in this section.	C
48	The <i>opendir()</i> function opens a directory stream corresponding to the directory named by	B
49	the <i>dirname</i> argument. The directory stream is positioned at the first entry.	C
50	If a file is removed from or added to the directory after the most recent call to <i>opendir()</i>	C
51	or <i>rewinddir()</i> , whether a subsequent call to <i>readdir()</i> returns an entry for that file is	C
52	unspecified.	C
53	The <i>readdir()</i> function returns a pointer to a structure representing the directory entry at	B
54	the current position in the directory stream to which <i>dirp</i> refers, and positions the	B
55	directory stream at the next entry. It returns a NULL pointer upon reaching the end of the	B
56	directory stream.	B
57	The <i>readdir()</i> function shall not return directory entries containing empty names. If	C
58	{_POSIX_DIR_DOTS} is in effect for <i>dirname</i> , entries for dot or dot-dot shall be	C
59	returned; otherwise they shall not be returned.	C
60	The pointer returned by <i>readdir()</i> points to data which may be overwritten by another	B
61	call to <i>readdir()</i> on the same directory stream. This data shall not be overwritten by	B
62	another call to <i>readdir()</i> on a different directory stream.	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

63	Upon successful completion, the <i>readdir()</i> function shall mark for update the <i>st_atime</i>	C
64	field of the directory.	C
65	The <i>rewinddir()</i> function resets the position of the directory stream to which <i>dirp</i> refers	B
66	to the beginning of the directory. It also causes the directory stream to refer to the	B
67	current state of the corresponding directory, as a call to <i>opendir()</i> would have done. It	B
68	does not return a value. If <i>dirp</i> does not refer to a directory stream, the effect is	C
69	undefined.	C
70	The <i>closedir()</i> function closes the directory stream referred to by <i>dirp</i> and returns a value	B
71	of zero if successful. Otherwise, it returns -1 indicating an error. Upon return, the value	C
72	of <i>dirp</i> may no longer point to an accessible object of type DIR .	C
73	5.1.2.3 Returns	
74	Upon successful completion, <i>opendir()</i> returns a pointer to an object of type DIR .	B
75	Otherwise, a value of NULL is returned and <i>errno</i> is set to indicate the error.	B
76	Upon successful completion, <i>readdir()</i> returns a pointer to an object of type <i>struct</i>	B
77	<i>dirent</i> . When an error is encountered, a value of NULL is returned and <i>errno</i> is set to	B
78	indicate the error. When the end of the directory is encountered, a value of NULL is	B
79	returned and <i>errno</i> is not changed.	B
80	Upon successful completion, <i>closedir()</i> returns a value of zero. Otherwise, a value of -1	B
81	is returned and <i>errno</i> is set to indicate the error.	B
82	5.1.2.4 Errors	B
83	If any of the following conditions occur, the <i>opendir()</i> function shall return -1 and set	B
84	<i>errno</i> to the corresponding value:	B
85	[EACCES] Search permission is denied for any component of <i>dirname</i> or read	C
86	permission is denied for <i>dirname</i> .	C
87	[EMFILE] Too many file descriptors are currently open for the process.	B
88	[ENOTDIR] A component of <i>dirname</i> is not a directory.	B
89		B
90	For each of the following conditions, if the condition is detected, the <i>readdir()</i> function	C
91	shall return -1 and set <i>errno</i> to the corresponding value:	C
92	[EBADF] The <i>dirp</i> argument does not refer to an open directory stream.	B
93		B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 94 For each of the following conditions, if the condition is detected, the *closedir()* function C
 95 shall return `-1` and set *errno* to the corresponding value: C
- 96 [EBADF] The *dirp* argument does not refer to an open directory stream. B
- 97 5.1.2.5 References
 98 `<dirent.h>` §5.1.1.
- 99 5.2 Working Directory
- 100 5.2.1 Change Current Working Directory
 101 Function: *chdir()*
- 102 5.2.1.1 Synopsis
- 103 `int chdir (path)` B
 104 `char *path;` B
- 105 5.2.1.2 Description
 106 The *path* argument points to the pathname of a directory. The *chdir()* function causes
 107 the named directory to become the current working directory, that is, the starting point
 108 for path searches of pathnames not beginning with slash. B
- 109 If the *chdir()* function fails, the current working directory shall remain unchanged. 9
- 110 5.2.1.3 Returns
 111 Upon successful completion, a value of zero is returned. Otherwise, a value of `-1` is
 112 returned and *errno* is set to indicate the error.
- 113 5.2.1.4 Errors
 114 If any of the following conditions occur, the *chdir()* function shall return `-1` and set B
 115 *errno* to the corresponding value: B
- 116 [EACCES] Search permission is denied for any component of the pathname.
- 117 [ENAMETOOLONG] C
 118 The *path* argument exceeds `{PATH_MAX}` in length, or a C
 119 pathname component is longer than `{NAME_MAX}` while C
 120 `{_POSIX_NO_TRUNC}` is in effect. C
- 121 [ENOTDIR] A component of the pathname is not a directory.
- 122 [ENOENT] The named directory does not exist or *path* is an empty string.

123 5.2.1.5 References

124 *getcwd()* §5.2.2.

125 5.2.2 Working Directory Pathname

126 Function: *getcwd()*

127 5.2.2.1 Synopsis

128 `char *getcwd (buf, size)` B129 `char *buf;` B130 `int size;` B

131 5.2.2.2 Description

132 The routine *getcwd()* copies the absolute pathname of the current working directory to A133 the character array pointed to by the argument *buf* and returns a pointer to the result. The134 *size* argument is the size in bytes of the character array pointed to by the *buf* argument. If A135 *buf* is a NULL pointer, the behavior of *getcwd()* is undefined. C

136 5.2.2.3 Returns

137 If successful, the *buf* argument is returned. A NULL pointer is returned if an error occurs138 and the variable *errno* is set to indicate the error. The contents of *buf* after an error is C

139 undefined. C

140 5.2.2.4 Errors

141 If any of the following conditions occur, the *getcwd()* function shall return -1 and set B142 *errno* to the corresponding value: B143 [EINVAL] The *size* argument is less than or equal to zero. C144 [ERANGE] The *size* argument is greater than zero, but is smaller than the A

145 length of the pathname.

146 For each of the following conditions, if the condition is detected, the *getcwd()* function C147 shall return -1 and set *errno* to the corresponding value: C

148 [EACCES] Read or search permission was denied for a component of the C

149 pathname. C

150 B

151 5.2.2.5 References

152 *chdir()* §5.2.1.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

153 5.3 General File Creation

154 5.3.1 Open a File

155 Function: *open()*

156 5.3.1.1 Synopsis

157 `#include <sys/types.h>` B158 `#include <fcntl.h>` C159 `int open (path, oflag, ...)`160 `char *path;`161 `int oflag;`

162 5.3.1.2 Description

163 The *open()* function establishes the connection between a file and a file descriptor. It C
 164 creates an open file description that refers to a file and a file descriptor that refers to that C
 165 open file description. The file descriptor is used by other I/O functions to refer to that A
 166 file. The *path* argument points to a pathname naming a file.

167 The *open()* function shall return a file descriptor for the named file which is the lowest B
 168 file descriptor not currently open for that process. The open file description is new, and C
 169 therefore the file descriptor does not share it with any other process in the system. The C
 170 file status flags and file access modes of the open file description shall be set according to
 171 the value of *oflag*. The value of *oflag* is the bitwise inclusive OR of values from the S
 172 following list. See <fcntl.h> §6.5.1 for the definitions of the symbolic constants. C
 173 Implementations may define additional flags, whose names shall begin with "O_." B
 174 Applications shall specify exactly one of the first three values (file access modes) below B
 175 in the value of *oflag*: B

176 `O_RDONLY` Open for reading only.177 `O_WRONLY` Open for writing only.178 `O_RDWR` Open for reading and writing.179 Any combination of the remaining flags may be specified in the value of *oflag*: A

180 `O_APPEND` If set, the file offset shall be set to the end of the file prior to C
 181 each write.

182 `O_CREAT` This option requires a third argument, *mode*, which is of type S
 183 *mode_t*. If the file exists, this flag has no effect. Otherwise, the
 184 file is created; the file's user ID shall be set to the process's B
 185 effective user ID; if `{_POSIX_GROUP_PARENT}` is in effect for C
 186 *path*, the file's group ID shall be set to the group ID of the C
 187 directory in which the file is being created; otherwise, the file's C
 188 group ID shall be set to the process's effective group ID. The C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

189		file permission bits (see <sys/stat.h> §5.6.1) shall be set to the	B
190		value of <i>mode</i> except those set in the process's file mode	
191		creation mask (see <i>umask()</i> §5.3.3). When bits in <i>mode</i> other	A
192		than the file permission bits are set, the effect is implementation	A
193		defined. The <i>mode</i> argument does not affect whether the file is	
194		opened for reading, for writing, or for both.	
195	O_EXCL	If O_EXCL and O_CREAT are set, <i>open()</i> shall fail if the file	
196		exists. If O_EXCL is set and O_CREAT is not set, the result is	A
197		implementation defined.	A
198	O_NONBLOCK		
199			A
200		When opening a FIFO with O_RDONLY or O_WRONLY set:	
201		If O_NONBLOCK is set:	
202		An <i>open()</i> for reading-only shall return without	
203		delay. An <i>open()</i> for writing-only shall return an	
204		error if no process currently has the file open for	
205		reading.	
206		If O_NONBLOCK is clear:	
207		An <i>open()</i> for reading-only shall block until a	
208		process opens the file for writing. An <i>open()</i> for	
209		writing-only shall block until a process opens the	
210		file for reading.	
211		When opening a block special or character special file that	B
212		supports nonblocking opens:	B
213		If O_NONBLOCK is set:	B
214		The <i>open()</i> shall return without waiting for the	B
215		device to be ready or available. Subsequent	B
216		behavior of the device is device specific.	B
217		If O_NONBLOCK is clear:	B
218		The <i>open()</i> shall wait until the device is ready or	B
219		available before returning.	B
220		Otherwise, the behavior of O_NONBLOCK is unspecified.	B
221	O_TRUNC	If the file exists and is a regular file, it shall be truncated to zero	B
222		length and the mode and owner shall be unchanged.	
223		If O_CREAT is set and the file did not previously exist, upon successful completion, the	C
224		<i>open()</i> function shall mark for update the <i>st_atime</i> , <i>st_ctime</i> , and <i>st_mtime</i> fields of the	C
225		file and the <i>st_ctime</i> and <i>st_mtime</i> fields of the parent directory.	C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 226 If `O_TRUNC` is set and the file did previously exist, upon successful completion, the `open()` function shall mark for update the `st_ctime` and `st_mtime` fields of the file. C
- 227 C
- 228 **5.3.1.3 Returns**
- 229 Upon successful completion, the function shall open the file and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, it shall C
- 230 return `-1` and shall set `errno` to indicate the error. No files shall be created or modified if C
- 231 the function returns `-1`. C
- 232 C
- 233 **5.3.1.4 Errors**
- 234 If any of the following conditions occur, the `open()` function shall return `-1` and set `errno` B
- 235 to the corresponding value: B
- 236 `[EACCES]` Search permission is denied on a component of the path prefix, or C
- 237 the file exists and the permissions specified by `oflag` are denied, or C
- 238 the file does not exist and write permission is denied for the parent C
- 239 directory of the file to be created. C
- 240 `[EEXIST]` `O_CREAT` and `O_EXCL` are set, and the named file exists.
- 241 `[EINTR]` The `open()` operation was terminated prematurely by a signal.
- 242 `[EISDIR]` The named file is a directory and the `oflag` argument specifies write A
- 243 or read/write access. A
- 244 `[EMFILE]` Too many file descriptors are currently in use by this process.
- 245 `[ENAMETOOLONG]` C
- 246 The length of the `path` string exceeds `{PATH_MAX}`, or a C
- 247 pathname component is longer than `{NAME_MAX}` while C
- 248 `{_POSIX_NO_TRUNC}` is in effect. C
- 249 `[ENFILE]` Too many files are currently open in the system.
- 250 `[ENOENT]` `O_CREAT` is not set and the named file does not exist; or C
- 251 `O_CREAT` is set and either the path prefix does not exist or the C
- 252 `path` argument points to an empty string. C
- 253 `[ENOSPC]` The directory or file system which would contain the new file C
- 254 cannot be extended. C
- 255 `[ENOTDIR]` A component of the path prefix is not a directory.
- 256 `[ENXIO]` `O_NONBLOCK` is set, the named file is a FIFO, `O_WRONLY` is set, C
- 257 and no process has the file open for reading. C
- 258 `[EROFS]` The named file resides on a read-only file system and either C
- 259 `O_WRONLY`, `O_RDWR`, or `O_CREAT` (if file does not exist) is set C
- 260 in the `oflag` argument. C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

261 5.3.1.5 References

262 *close()* §6.3.1, *creat()* §5.3.2, *dup()* §6.2.1, *exec* §3.1.2, *fcntl()* §6.5.2, <fcntl.h> §6.5.1, B
 263 *lseek()* §6.5.3, *read()* §6.4.1, *sigaction()* §3.3.4, *stat()* §5.6.2, <sys/stat.h> §5.6.1, B
 264 *write()* §6.4.2, *umask()* §5.3.3.

265 5.3.2 Create a New File or Rewrite an Existing One

266 Function: *creat()*

267 5.3.2.1 Synopsis

268 `#include <sys/types.h>`269 `int creat (path, mode)` C270 `char *path;`271 `mode_t mode;` 8

272 5.3.2.2 Description

273

274 The function call

275 `creat (path, mode);`

276 is equivalent to

277 `open (path, O_WRONLY | O_CREAT | O_TRUNC, mode);`

278 9

279 5.3.2.3 References

280 *open()* §5.3.1, <sys/stat.h> §5.6.1. B

281 5.3.3 Set File Creation Mask

282 Function: *umask()*

283 5.3.3.1 Synopsis

284 `#include <sys/types.h>`285 `mode_t umask (cmask)` C286 `mode_t cmask;` 8

287 5.3.3.2 Description

288 The *umask()* routine sets the process's file mode creation mask to *cmask* and returns the
 289 previous value of the mask. Only the file permission bits (see <sys/stat.h> §5.6.1) of 8
 290 *cmask* are used. 8

291 The process's file mode creation mask is used during *open()*, *creat()*, *mkdir()*, and C
 292 *mkfifo()* functions to turn off permission bits in the *mode* argument supplied. Bit B
 293 positions that are set in *cmask* are cleared in the mode of the created file. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

294 5.3.3.3 Returns

295 The previous value of the file mode creation mask is returned.

296 5.3.3.4 References

297 *chmod()* §5.6.4, *creat()* §5.3.2, *mkdir()* §5.4.1, *mkfifo()* §5.4.2, *open()* §5.3.1,
298 *<signal.h>* §3.3.1, *<sys/stat.h>* §5.6.1.A
B

299 5.3.4 Link to a File

300 Function: *link()*

301 5.3.4.1 Synopsis

302 `int link(path1, path2)`
303 `char *path1, *path2;`

304 5.3.4.2 Description

305 The argument *path1* points to a pathname naming an existing file. The argument *path2*
306 points to a pathname naming the new directory entry to be created. The *link()* function
307 shall create a new link for the existing file. The link count of the file is incremented by
308 one.8
8309 If the *link()* function fails, no link shall be created.

9

310 If *path1* names a directory, the effect of this function is dependent on the definition of
311 `{_POSIX_LINK_DIR}`. If in effect for *path1*, the link is created, subject to any other
312 restrictions listed for the function; otherwise, the linking of a directory shall be
313 disallowed and the function shall fail.c
c
c
c314 Upon successful completion, the *link()* function shall mark for update the *st_ctime* field
315 of the file. Also, the *st_ctime* and *st_mtime* fields of the directory that contains the new
316 entry are marked for update.c
c
c

317 5.3.4.3 Returns

318 Upon successful completion, *link()* shall return a value of zero. Otherwise, a value of -1
319 is returned and *errno* is set to indicate the error.

320 5.3.4.4 Errors

321 If any of the following conditions occur, the *link()* function shall return -1 and set *errno*
322 to the corresponding value:B
B323 `[EACCES]` A component of either path prefix denies search permission, or the
324 requested link requires writing in a directory with a mode that
325 denies write permission.326 `[EEXIST]` The link named by *path2* exists.327 `[EMLINK]` The number of links to the file named by *path1* would exceed
328 `{LINK_MAX}`.

329	[ENAMETOOLONG]		C
330		The length of the <i>path1</i> or <i>path2</i> string exceeds {PATH_MAX}, or	C
331		a pathname component is longer than {NAME_MAX} while	C
332		{_POSIX_NO_TRUNC} is in effect.	C
333	[ENOENT]	A component of either path prefix does not exist; the file named by	
334		<i>path1</i> does not exist; or either <i>path1</i> or <i>path2</i> points to an empty	
335		string.	
336	[ENOSPC]	The directory that would contain the link cannot be extended.	
337	[ENOTDIR]	A component of either path prefix is not a directory.	
338	[EPERM]	The file named by <i>path1</i> is a directory and the implementation	B
339		restricts the linking of directories to processes with appropriate	B
340		privileges, and the calling process does not have appropriate	B
341		privileges.	B
342	[EROFS]	The requested link requires writing in a directory on a read-only	
343		file system.	
344	[EXDEV]	The link named by <i>path2</i> and the file named by <i>path1</i> are on	A
345		different file systems and the implementation does not support	A
346		links between file systems.	A
347	5.3.4.5 References		8
348	<i>rename()</i> §5.5.3, <i>unlink()</i> §5.5.1.		8
349	5.4 Special File Creation		
350	5.4.1 Make a Directory		
351	Function: <i>mkdir()</i>		
352	5.4.1.1 Synopsis		
353		#include <sys/types.h>	C
354		int mkdir (<i>path</i> , <i>mode</i>)	
355		char * <i>path</i> ;	
356		mode_t <i>mode</i> ;	8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

357 5.4.1.2 Description

358 The *mkdir()* routine creates a new directory with name *path*. The file permission bits of c
 359 the new directory are initialized from *mode*. The file permission bits of the *mode* 9
 360 argument are modified by the process's file creation mask (see *umask()* §5.3.3). When A
 361 bits in *mode* other than the file permission bits are set, the effect is implementation A
 362 defined. A

363 The directory's owner ID is set to the process's effective user ID. If c
 364 `{_POSIX_GROUP_PARENT}` is in effect for *path*, the directory's group ID shall be set to c
 365 the group ID of the directory in which the directory is being created; otherwise, the c
 366 directory's group ID shall be set to the process's effective group ID. c

367 If `{_POSIX_DIR_DOTS}` is in effect for *path*, the newly created directory shall contain c
 368 only entries for dot and dot-dot; otherwise the directory shall be empty. c

369 Upon successful completion, the *mkdir()* function shall mark for update the *st_atime*, c
 370 *st_ctime*, and *st_mtime* fields of the directory. Also, the *st_ctime* and *st_mtime* fields of c
 371 the directory that contains the new entry are marked for update. c

372 5.4.1.3 Returns

373 A return value of zero indicates success. A return value of -1 indicates that an error has
 374 occurred and an error code is stored in *errno*. No directory shall be created if the return
 375 value is -1.

376 5.4.1.4 Errors

377 If any of the following conditions occur, the *mkdir()* function shall return -1 and set B
 378 *errno* to the corresponding value: B

379 [EACCES] Search permission is denied on a component of the path prefix, or
 380 write permission is denied on the parent directory of the directory
 381 to be created.

382 [EEXIST] The named file exists.

383 [EMLINK] The link count of the parent directory would exceed s
 384 `{LINK_MAX}`.

385 [ENAMETOOLONG] c
 386 The length of the *path* argument exceeds `{PATH_MAX}`, or a c
 387 pathname component is longer than `{NAME_MAX}` while c
 388 `{_POSIX_NO_TRUNC}` is in effect. c

389 [ENOENT] A component of the path prefix does not exist or the *path* argument
 390 points to an empty string.

391 [ENOSPC] The file system does not contain enough space to hold the contents
 392 of the new directory or to extend the parent directory of the new
 393 directory.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

394 [ENOTDIR] A component of the path prefix is not a directory.

395 [EROFS] The path prefix resides on a read-only file system.

396 5.4.1.5 References

397 *chmod()* §5.6.4, *stat()* §5.6.2, *<sys/stat.h>* §5.6.1, *umask()* §5.3.3. B

398 5.4.2 Make a FIFO Special File

399 Function: *mkfifo()*

400 5.4.2.1 Synopsis

401 #include *<sys/types.h>* B

402 int *mkfifo(path, mode)*

403 char **path*;

404 mode_t *mode*; 8

405 5.4.2.2 Description

406 The *mkfifo()* routine creates a new FIFO special file named by the pathname pointed to by

407 *path*. The mode of the new FIFO is initialized from *mode*. The file permission bits of 9

408 the *mode* argument are modified by the process's file creation mask (see *umask()* §5.3.3). A

409 When bits in *mode* other than the file permission bits are set, the effect is implementation A

410 defined. A

411 The FIFO's owner ID shall be set to the process's effective user ID. If c

412 {_POSIX_GROUP_PARENT} is in effect for *path*, the FIFO's group ID shall be set to the c

413 group ID of the directory in which the FIFO is being created; otherwise, the FIFO's group c

414 ID shall be set to the process's effective group ID. c

415 Upon successful completion, the *mkfifo()* function shall mark for update the *st_atime*, c

416 *st_ctime*, and *st_mtime* fields of the file. Also, the *st_ctime* and *st_mtime* fields of the c

417 directory that contains the new entry are marked for update. c

418 5.4.2.3 Returns

419 Upon successful completion a value of zero is returned. Otherwise, a value of -1 is

420 returned, no FIFO is created, and *errno* is set to indicate the error.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

421 5.4.2.4 Errors

422 If any of the following conditions occur, the *mkfifo()* function shall return `-1` and set `errno` to the corresponding value: B

423 `[EACCES]` A component of the path prefix denies search permission. B

424 `[EEXIST]` The named file already exists.

425 `[ENAMETOOLONG]` C

426 The length of the *path* string exceeds `{PATH_MAX}`, or a C
 427 pathname component is longer than `{NAME_MAX}` while C
 428 `{_POSIX_NO_TRUNC}` is in effect. C

429 `[ENOENT]` A component of the path prefix does not exist or the *path* argument C
 430 points to an empty string.

431 `[ENOSPC]` The directory that would contain the new file cannot be extended C
 432 or the file system is out of file allocation resources.

433 `[ENOTDIR]` A component of the path prefix is not a directory.

434 `[EROFS]` The named file resides on a read-only file system.

435 5.4.2.5 References

436 *chmod()* §5.6.4, *exec* §3.1.2, *pipe()* §6.1.1, *stat()* §5.6.2, `<sys/stat.h>` §5.6.1, *umask()* B
 437 §5.3.3.

438 5.5 File Removal

439 5.5.1 Remove Directory Entries

440 Function: *unlink()*

441 5.5.1.1 Synopsis

442 `int unlink (path)`

443 `char *path;`

444 5.5.1.2 Description

445 The *unlink()* function shall remove the link named by the pathname pointed to by *path* B
 446 and decrement the link count of the file referenced by the link. B

447 When the file's link count becomes zero and no process has the file open, the space C
 448 occupied by the file shall be freed and the file shall no longer be accessible. If one or C
 449 more processes have the file open when the last link is removed, the removal shall be C
 450 postponed until all references to the file have been closed.

451 If *path* names a directory, the effect of this function is dependent on the definition of C
 452 `{_POSIX_LINK_DIR}`. If in effect for *path*, the link is removed, subject to any other C
 453 restrictions listed for the function; otherwise, the unlinking of a directory shall be C
 454

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 455 disallowed and the function shall fail. Applications should use *rmdir()* to remove a
456 directory.
- 457 Upon successful completion, the *unlink()* function shall mark for update the *st_ctime* and c
458 *st_mtime* fields of the parent directory. Also, if the file's link count is not zero, the c
459 *st_ctime* field of the file shall be marked for update. c
- 460 **5.5.1.3 Returns**
- 461 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 c
462 shall be returned and *errno* shall be set to indicate the error. If -1 is returned, the named c
463 file shall not be changed. c
- 464 **5.5.1.4 Errors**
- 465 If any of the following conditions occur, the *unlink()* function shall return -1 and set B
466 *errno* to the corresponding value: B
- 467 [EACCES] Search permission is denied for a component of the path prefix, or
468 write permission is denied on the directory containing the link to
469 be removed.
- 470 [ENAMETOOLONG] c
471 The length of the *path* argument exceeds {PATH_MAX}, or a c
472 pathname component is longer than {NAME_MAX} while c
473 {_POSIX_NO_TRUNC} is in effect. c
- 474 [ENOENT] The named file does not exist or the *path* argument points to an
475 empty string.
- 476 [ENOTDIR] A component of the path prefix is not a directory.
- 477 [EROFS] The directory entry to be unlinked is part of a read-only file
478 system.
- 479 For each of the following conditions, if the condition is detected, the *unlink()* function B
480 shall return -1 and set *errno* to the corresponding value: B
- 481 [EBUSY] The file named by the *path* argument cannot be unlinked because it
482 is being used by the system or another process.
- 483 [EPERM] The named file is a directory and the implementation restricts the B
484 unlinking of directories to processes with appropriate privileges, B
485 and the calling process does not have appropriate privileges. B

486 5.5.1.5 References

487 *close()* §6.3.1, *link()* §5.3.4, *open()* §5.3.1, *rename()* §5.5.3, *rmdir()* §5.5.2. 8

488 5.5.2 Remove a Directory

489 Function: *rmdir()*

490 5.5.2.1 Synopsis

```
491         int rmdir (path)
492         char *path;
```

493 5.5.2.2 Description

494 The *rmdir()* function removes a directory whose name is given by *path*. If c
 495 `{_POSIX_DIR_DOTS}` is in effect for *path*, the directory shall be removed only if there c
 496 are no entries other than dot or dot-dot; otherwise the directory shall be removed only if c
 497 it has no entries. c

498 If the directory is the root directory or the current working directory, the effect of this c
 499 function is implementation defined. c

500 Upon successful completion, the *rmdir()* function shall mark for update the *st_ctime* and c
 501 *st_mtime* fields of the parent directory. c

502 5.5.2.3 Returns

503 A return value of zero indicates success. A return value of `-1` indicates that an error has c
 504 occurred and an error code has been stored in *errno*. c

505 5.5.2.4 Errors

506 If any of the following conditions occur, the *rmdir()* function shall return `-1` and set B
 507 *errno* to the corresponding value: B

508 [EACCES] Search permission is denied on a component of the path or write
 509 permission is denied on the parent directory of the directory to be
 510 removed.

511 [EEXIST] or [ENOTEMPTY] B
 512 The *path* argument names a directory containing files other than B
 513 dot and dot-dot. B

514 [ENAMETOOLONG] c
 515 The length of the *path* argument exceeds `{PATH_MAX}`, or a c
 516 pathname component is longer than `{NAME_MAX}` while c
 517 `{_POSIX_NO_TRUNC}` is in effect. c

518 [ENOENT] The *path* argument names a non-existent directory or points to an
 519 empty string.

520 [ENOTDIR] A component of the path is not a directory.

- 521 [EROFS] The directory entry to be removed resides on a read-only file
522 system.
- 523 For each of the following conditions, if the condition is detected, the *rmdir()* function B
524 shall return -1 and set *errno* to the corresponding value: B
- 525 [EBUSY] The directory to be removed is currently in use by the system or
526 another process.
- 527 5.5.2.5 References
528 *mkdir()* §5.4.1, *unlink()* §5.5.1.
- 529 5.5.3 Rename a File
530 Function: *rename()*
- 531 5.5.3.1 Synopsis
- 532 `int rename (old, new)`
533 `char *old;`
534 `char *new;`
- 535 5.5.3.2 Description
536 The *rename()* function changes the name of a file. The *old* argument points to the
537 pathname of the file to be renamed. The *new* argument points to the new pathname of the
538 file.
- 539 If the *old* argument and the *new* argument both refer to links to the same existing file, the c
540 *rename()* function shall return successfully and perform no other action. c
- 541 If the *old* argument points to the pathname of a file that is not a directory, the *new* B
542 argument shall not point to the pathname of a directory. If the link named by the *new* B
543 argument exists, it shall be removed and *old* renamed to *new*. In this case, c
544 implementations shall ensure that a link named *new* remains visible to other processes c
545 throughout the renaming operation. Write access permission is required for both the
546 directory containing *old* and the directory containing *new*.
- 547 If the *old* argument points to the pathname of a directory, the *new* argument shall not B
548 point to the pathname of a file that is not a directory. If the directory named by the *new* B
549 argument exists, it shall be removed and *old* renamed to *new*. In this case, c
550 implementations shall ensure that a link named *new* remains visible to other processes c
551 throughout the renaming operation. Thus, if *new* names an existing directory, the c
552 directory shall be required to have only the entries *dot* and *dot-dot*, if c
553 `{_POSIX_DIR_DOTS}` is in effect for *new*; if `{_POSIX_DIR_DOTS}` is not in effect, the c
554 existing directory shall be required to be empty. The *new* pathname shall not name a c
555 descendant of *old*. Write access permission is required for the directory containing *old* c
556 and the directory containing *new*. If the *old* argument points to the pathname of a c
557 directory, write access permission may be required for the directory named by *old*, and, c
558 if it exists, the directory named by *new*. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

559	Upon successful completion, the <i>rename()</i> function shall mark for update the <i>st_ctime</i>	C
560	and <i>st_mtime</i> fields of the parent directory of each file.	C
561	5.5.3.3 Returns	
562	A return value of zero indicates success. A return value of -1 indicates that an error has	
563	occurred and an error code has been stored in <i>errno</i> .	
564	5.5.3.4 Errors	
565	If any of the following conditions occur, the <i>rename()</i> function shall return -1 and set	B
566	<i>errno</i> to the corresponding value:	B
567	[EACCES] A component of either path prefix denies search permission; or one	
568	of the directories containing <i>old</i> or <i>new</i> denies write permissions;	C
569	or, write permission is required and is denied for a directory	C
570	pointed to by the <i>old</i> or <i>new</i> arguments.	C
571	[EEXIST] or [ENOTEMPTY]	B
572	The link named by <i>new</i> is a directory containing entries other than	
573	dot and dot-dot.	
574	[EINVAL] The <i>new</i> directory is an ancestor or a descendant of the <i>old</i>	A
575	directory.	A
576	[EISDIR] The <i>new</i> argument points to a directory and the <i>old</i> argument	8
577	points to a file that is not a directory.	8
578	[ENAMETOOLONG]	C
579	The length of the <i>old</i> or <i>new</i> argument exceeds {PATH_MAX}, or	C
580	a pathname component is longer than {NAME_MAX} while	C
581	{_POSIX_NO_TRUNC} is in effect.	C
582	[ENOENT] The link named by the <i>old</i> argument does not exist or either <i>old</i> or	
583	<i>new</i> points to an empty string.	
584	[ENOSPC] The directory that would contain <i>new</i> cannot be extended.	
585	[ENOTDIR] A component of either path prefix is not a directory; or the <i>old</i>	
586	argument names a directory and the <i>new</i> argument names a	
587	nondirectory file.	
588		C
589	[EROFS] The requested operation requires writing in a directory on a read-	
590	only file system.	

591 For each of the following conditions, if the condition is detected, the *rename()* function B
592 shall return -1 and set *errno* to the corresponding value: B

593 [EBUSY] The link named by *old* or *new* is currently in use by the system or
594 another process.

595 B

596 [EXDEV] The links named by *new* and *old* are on different file systems.

597 5.5.3.5 References

598 *link()* §5.3.4, *rmdir()* §5.5.2, *unlink()* §5.5.1.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

599 5.6 File Characteristics

600 5.6.1 File Characteristics: Header File and Data Structure

601 5.6.1.1 Synopsis

602 #include <sys/types.h>

603 #include <sys/stat.h>

B

604 5.6.1.2 Description

605 The header <sys/stat.h> defines the structure *stat* returned by the functions *stat()* and606 *fstat()*.

B

<u>Member Type</u>	<u>Member Name</u>	<u>Description</u>	
<i>mode_t</i>	<i>st_mode</i>	File mode (see list below)	B
<i>ino_t</i>	<i>st_ino</i>	File serial number	
<i>dev_t</i>	<i>st_dev</i>	ID of device containing a directory entry for this file.	
		File serial number and device ID taken together uniquely identify the file within the system.	
<i>dev_t</i>	<i>st_rdev</i>	ID of device. This entry is valid only for character special or block special files.	
<i>nlink_t</i>	<i>st_nlink</i>	Number of links	
<i>uid_t</i>	<i>st_uid</i>	User ID of the file's owner	B
<i>uid_t</i>	<i>st_gid</i>	Group ID of the file's group	B
<i>off_t</i>	<i>st_size</i>	For regular files, this is the file size in bytes. For other file types, the use of this field is unspecified.	B
<i>time_t</i>	<i>st_atime</i>	Time of last access	B
<i>time_t</i>	<i>st_mtime</i>	Time of last data modification	
<i>time_t</i>	<i>st_ctime</i>	Time of last file status change	

628 All of the described members must appear in the *stat* structure. The *stat* structure may629 also include other data elements as well. The structure members *st_mode*, *st_ino*,630 *st_dev*, *st_uid*, *st_gid*, and *st_mtime* shall have meaningful values for all file types631 defined in this standard. The value of the member *st_rdev* is implementation defined.632 The value of the member *st_nlink* shall be set to the number of links to the file.

633 5.6.1.2.1 <sys/stat.h> File Types

B

634 The following macros shall test whether a file is of the specified type. The value *m*635 supplied to the macros is the value of *st_mode* from a *struct stat*. The macro evaluates to

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

636		a non-zero value if the test is true, zero if the test is false.	
637	<code>S_ISDIR(m)</code>	Test macro for directory file	
638	<code>S_ISCHR(m)</code>	Test macro for character special file	
639	<code>S_ISBLK(m)</code>	Test macro for block special file	
640	<code>S_ISREG(m)</code>	Test macro for regular file	
641	<code>S_ISFIFO(m)</code>	Test macro for FIFO special file	
642	5.6.1.2.2 <sys/stat.h> File Modes		c
643	The <code>st_mode</code> value is bit-encoded with the following masks and bits:		c
644	<code>S_IRWXU</code>	Read, write, search (if a directory), or execute (otherwise)	c
645		permissions mask for the file owner class.	c
646	<code>S_IRUSR</code>	Read permission bit for the file owner class.	c
647	<code>S_IWUSR</code>	Write permission bit for the file owner class.	c
648	<code>S_IXUSR</code>	Search (if a directory) or execute (otherwise)	c
649		permissions bit for the file owner class.	c
650	<code>S_IRWXG</code>	Read, write, search (if a directory), or execute (otherwise)	c
651		permissions mask for the file group class.	c
652	<code>S_IRGRP</code>	Read permission bit for the file group class.	c
653	<code>S_IWGRP</code>	Write permission bit for the file group class.	c
654	<code>S_IXGRP</code>	Search (if a directory) or execute (otherwise)	c
655		permissions bit for the file group class.	c
656	<code>S_IRWXO</code>	Read, write, search (if a directory), or execute (otherwise)	c
657		permissions mask for the file other class.	c
658	<code>S_IROTH</code>	Read permission bit for the file other class.	c
659	<code>S_IWOTH</code>	Write permission bit for the file other class.	c
660	<code>S_IXOTH</code>	Search (if a directory) or execute (otherwise)	c
661		permissions bit for the file other class.	c
662	<code>S_ISUID</code>	Set user ID on execution. The process's effective user ID shall be	
663		set to that of the owner of the file when the file is run as a program	
664		(see <i>exec</i>). This bit should be cleared on any write to the file.	A
665	<code>S_ISGID</code>	Set group ID on execution. Set effective group ID on the process	
666		to the file's group when the file is run as a program (see <i>exec</i>).	A
667		This bit should be cleared on any write to the file.	A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

668 The file permission bits are defined to be those corresponding to the bitwise inclusive 8
669 OR of `S_IRWXU`, `S_IRWXG`, and `S_IRWXO`. 8

670 **5.6.1.2.3 <sys/stat.h> Time Entries** B

671 The time-related fields of *struct stat* are as follows:

672 `st_atime` Accessed file data, e.g. *read()*. C

673 `st_mtime` Modified file data, e.g. *write()*. C

674 `st_ctime` Changed file status, e.g. *chmod()*. C

675 These times are updated as described by file times update §2.4. C

676 All the functions in this standard that change these fields directly describe those changes C
677 in the context of the functions' definitions. Other functions that directly change *st_atime*, C
678 *st_mtime*, or *st_ctime* shall be implementation defined. C

679 Times are given in seconds since the Epoch (see Epoch §2.3). C

680 **5.6.1.3 References** C

681 *chmod()* §5.6.4, *chown()* §5.6.5, *creat()* §5.3.2, *link()* §5.3.4, *mkdir()* §5.4.1, *mkfifo()* B
682 §5.4.2, *pipe()* §6.1.1, *read()* §6.4.1, *unlink()* §5.5.1, *utime()* §5.6.6, *write()* §6.4.2, 8
683 *remove()* (ANSI/X3.159-198x Programming Language C Standard).

684 **5.6.2 Get File Status**

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

686 **5.6.2.1 Synopsis**

687 #include <sys/types.h>
688 #include <sys/stat.h> B

689 int stat (*path*, *buf*)
690 char **path*;
691 struct stat **buf*;

692 int fstat (*fdes*, *buf*)
693 int *fdes*;
694 struct stat **buf*;

695 **5.6.2.2 Description**

696 The *path* argument points to a pathname naming a file. Read, write or execute
697 permission for the named file is not required, but all directories listed in the pathname
698 leading to the file must be searchable. The *stat()* function obtains information about the
699 named file and writes it to the area pointed to by the *buf* argument.

700 Similarly, the *fstat()* function obtains information about an open file known by the file
701 descriptor *fdes*.

- 702 Additional implementation defined access constraints may cause the *stat()* and *fstat()* c
 703 functions to fail. c
- 704 Both functions update any time-related fields as described in file times update §2.4 c
 705 before writing into the *stat* structure. c
- 706 The *buf* is taken to be a pointer to a *stat* structure, as defined in the header <sys/stat.h> B
 707 §5.6.1, into which information is placed concerning the file.
- 708 **5.6.2.3 Returns**
- 709 Upon successful completion a value of zero shall be returned. Otherwise, a value of -1 c
 710 shall be returned and *errno* shall be set to indicate the error. c
- 711 **5.6.2.4 Errors**
- 712 If any of the following conditions occur, the *stat()* function shall return -1 and set *errno* B
 713 to the corresponding value: B
- 714 [EACCES] Search permission is denied for a component of the path prefix. c
- 715 [ENAMETOOLONG] c
 716 The length of the *path* argument exceeds {PATH_MAX}, or a c
 717 pathname component is longer than {NAME_MAX} while c
 718 {_POSIX_NO_TRUNC} is in effect. c
- 719 [ENOENT] The named file does not exist or the *path* argument points to an
 720 empty string.
- 721 [ENOTDIR] A component of the path prefix is not a directory.
- 722 If any of the following conditions occur, the *fstat()* function shall return -1 and set *errno* B
 723 to the corresponding value: B
- 724 [EBADF] The *fdes* argument is not a valid file descriptor.
- 725 B
- 726 **5.6.2.5 References**
- 727 *creat()* §5.3.2, *dup()* §6.2.1, *fcntl()* §6.5.2, *open()* §5.3.1, *pipe()* §6.1.1, <sys/stat.h> B
 728 §5.6.1.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

729 5.6.3 File Accessibility

730 Function: *access()* B

731 5.6.3.1 Synopsis

732 #include <unistd.h>

733 int *access* (*path*, *amode*)734 char **path*;735 int *amode*;

736 B

737 5.6.3.2 Description

738 The *access()* function checks the accessibility of the file named by the pathname pointed C739 to by the *path* argument for the file access permissions indicated by *amode*, using the real C

740 user ID in place of the effective user ID and the real group ID in place of the effective C

741 group ID. C

742 The value of *amode* is either the bitwise inclusive OR of the access permissions to be C

743 checked. (R_OK, W_OK, and X_OK) or the existence test, F_OK. See Symbolic C

744 Constants §2.10 for the description of these symbolic constants. C

745 If any access permission is to be checked, each shall be checked individually, as C

746 described in file access permissions §2.4. If the process has appropriate privileges, an C

747 implementation may substitute search permissions for execute permission. C

748 5.6.3.3 Returns

749 If the requested access is permitted, a value of zero shall be returned. Otherwise, a value C

750 of -1 shall be returned and *errno* shall be set to indicate the error. C

751 5.6.3.4 Errors

752 If any of the following conditions occur, the *access()* function shall return -1 and set B753 *errno* to the corresponding value: B754 [EACCES] The permissions specified by *amode* are denied, or search A
755 permission is denied on a component of the path prefix. A756 [ENAMETOOLONG] C
757 The length of the *path* argument exceeds {PATH_MAX}, or a C
758 pathname component is longer than {NAME_MAX} while C
759 {_POSIX_NO_TRUNC} is in effect. C760 [ENOENT] The *path* argument points to an empty string or to the name of a A
761 file that does not exist. A

762 [ENOTDIR] A component of the path prefix is not a directory. A

763 [EROFS] Write access requested for a file on a read-only file system. A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

764	For each of the following conditions, if the condition is detected, the <i>access()</i> function	c
765	shall return -1 and set <i>errno</i> to the corresponding value:	c
766	[EINVAL] Invalid value specified for <i>amode</i> .	c
767	5.6.3.5 References	8
768	<i>chmod()</i> §5.6.4, <i>stat()</i> §5.6.2, <unistd.h> §2.10.	8
769	5.6.4 Change File Modes	8
770	Function: <i>chmod()</i>	8
771	5.6.4.1 Synopsis	8
772	#include <sys/types.h>	8
773	#include <sys/stat.h>	B
774	int <i>chmod</i> (<i>path</i> , <i>mode</i>)	8
775	char * <i>path</i> ;	8
776	mode_t <i>mode</i> ;	8
777	5.6.4.2 Description	8
778	The <i>path</i> argument shall point to a pathname naming a file. If the effective user ID of the	c
779	calling process matches the file owner or has appropriate privileges, the <i>chmod()</i>	c
780	function shall set the file mode, as described in <sys/stat.h> §5.6.1, of the named file	c
781	from the corresponding bits in the <i>mode</i> argument. These bits define access permissions	c
782	for the user associated with the file, the group associated with the file, and all others, as	c
783	described in file access permissions §2.4. Additional implementation defined	c
784	restrictions may cause the S_ISUID and S_ISGID bits in <i>mode</i> to be ignored.	c
785	If the calling process does not have appropriate privileges, and if the group ID of the file	8
786	does not match the effective group ID or one of the supplementary group IDs, bit S_ISGID	8
787	(set group ID on execution) in the file's mode shall be cleared upon successful return	c
788	from <i>chmod()</i> .	c
789	The effect on file descriptors for files open at the time of the <i>chmod()</i> function is	8
790	implementation defined.	8
791	Upon successful completion, the <i>chmod()</i> function shall mark for update the <i>st_ctime</i>	c
792	field of the file.	c
793	5.6.4.3 Returns	8
794	Upon successful completion, the function shall return a value of zero. Otherwise, a value	8
795	of -1 shall be returned and <i>errno</i> shall be set to indicate the error. If -1 is returned, no	8
796	change to the file mode shall have occurred.	8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

797	5.6.4.4 Errors		B
798	If any of the following conditions occur, the <i>chmod()</i> function shall return <i>-1</i> and set		B
799	<i>errno</i> to the corresponding value:		B
800	[EACCES] Search permission is denied on a component of the path prefix.		
801	[ENAMETOOLONG]		C
802	The length of the <i>path</i> argument exceeds {PATH_MAX}, or a		C
803	pathname component is longer than {NAME_MAX} while		C
804	{_POSIX_NO_TRUNC} is in effect.		C
805	[ENOTDIR] A component of the path prefix is not a directory.		
806	[ENOENT] The named file does not exist or the <i>path</i> argument points to an		
807	empty string.		
808	[EPERM] The effective user ID does not match the owner of the file and the		B
809	calling process does not have the appropriate privileges.		B
810	[EROFS] The named file resides on a read-only file system.		
811	5.6.4.5 References		
812	<i>chown()</i> §5.6.5, <i>mkdir()</i> §5.4.1, <i>mkfifo()</i> §5.4.2, <i>stat()</i> §5.6.2, <sys/stat.h> §5.6.1.		B
813	5.6.5 Change Owner and Group of a File		
814	Function: <i>chown()</i>		
815	5.6.5.1 Synopsis		
816	#include <sys/types.h>		B
817	int <i>chown</i> (<i>path</i> , <i>owner</i> , <i>group</i>)		
818	char * <i>path</i> ;		
819	uid_t <i>owner</i> , <i>group</i> ;		B
820	5.6.5.2 Description		
821	The <i>path</i> argument points to a pathname naming a file. The user ID and group ID of the		
822	named file are set to the numeric values contained in <i>owner</i> and <i>group</i> respectively.		
823	Only processes with an effective user ID equal to the user ID of the file or with		B
824	appropriate privileges may change the ownership of a file. If		C
825	{_POSIX_CHOWN_RESTRICTED} is in effect for <i>path</i> , this operation is restricted to		C
826	processes with appropriate privileges. If {_POSIX_CHOWN_SUP_GRP} is in effect for		C
827	<i>path</i> , the implementation limits a process with an effective user ID equal to the user ID of		B
828	the file, but without appropriate privileges, to changing the group ID of a file only to the		B
829	effective group ID of the process or to one of the supplementary group IDs.		B
830	The set-user-ID (S_ISUID) and set-group-ID (S_ISGID) bits of the file mode shall be		C
831	cleared upon successful return from <i>chown()</i> , unless the the call is made by a process		C
832	with appropriate privilege, in which case it is implementation defined whether those bits		C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

- 833 are altered. If the *chown()* function is successfully invoked on a file that is not a regular
 834 file, these bits may be cleared. These bits are defined in `<sys/stat.h>` §5.6.1.
- 835 Upon successful completion, the *chown()* function shall mark for update the *st_ctime* c
 836 field of the file. c
- 837 **5.6.5.3 Returns**
- 838 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 c
 839 shall be returned and *errno* shall be set to indicate the error. If -1 is returned, no change c
 840 shall be made in the owner and group of the file. c
- 841 **5.6.5.4 Errors**
- 842 If any of the following conditions occur, the *chown()* function shall return -1 and set B
 843 *errno* to the corresponding value: B
- 844 [EACCES] Search permission is denied on a component of the path prefix.
- 845 [ENAMETOOLONG] c
 846 The length of the *path* argument exceeds {PATH_MAX}, or a c
 847 pathname component is longer than {NAME_MAX} while c
 848 {_POSIX_NO_TRUNC} is in effect. c
- 849 [ENOTDIR] A component of the path prefix is not a directory.
- 850 [ENOENT] The named file does not exist or the *path* argument points to an
 851 empty string.
- 852 [EPERM] The effective user ID does not match the owner of the file or the B
 853 calling process does not have appropriate privileges. B
- 854 [EROFS] The named file resides on a read-only file system.
- 855 For each of the following conditions, if the condition is detected, the *chmod()* function c
 856 shall return -1 and set *errno* to the corresponding value: c
- 857 [EINVAL] The owner or group ID supplied is outside the range of zero to c
 858 {UID_MAX}, inclusive. c
- 859 **5.6.5.5 References**
- 860 *chmod()* §5.6.4, `<sys/stat.h>` §5.6.1. B

861 5.6.6 Set File Access and Modification Times

862 Function: *utime()*

863 5.6.6.1 Synopsis

864 `#include <sys/types.h>`865 `#include <utime.h>`866 `int utime (path, times)`867 `char *path;`868 `struct utimbuf *times;`

869 5.6.6.2 Description

870 The argument *path* points to a pathname naming a file. The *utime()* function sets the
871 access and modification times of the named file.872 If the *times* argument is NULL, the access and modification times of the file are set to the
873 current time. The effective user ID of the process must match the owner of the file, or the
874 process must have write permission or appropriate privilege, to use the *utime()* function
875 in this manner. c
c
c876 If *times* is not NULL, *times* is interpreted as a pointer to a *utimbuf* structure and the
877 access and modification times are set to the values contained in the designated structure. c
878 If `{_POSIX_UTIME_OWNER}` is in effect for *path*, the owner of the file shall be c
879 permitted to use the *utime()* function in this way, otherwise such use shall be restricted to B
880 processes with appropriate privileges. B881 The *utimbuf* structure is defined by the header `<utime.h>`, and includes the following
882 members:

<u>Member Type</u>	<u>Member Name</u>	<u>Description</u>
<i>time_t</i>	actime	Access time
<i>time_t</i>	modtime	Modification time

887 The times in the *utimbuf* structure are measured in seconds since the Epoch (see Epoch c
888 §2.3). c889 Upon successful completion, the *utime()* function shall mark for update the *st_ctime* field c
890 of the file. c

891 5.6.6.3 Returns

892 Upon successful completion, the function shall return a value of zero. Otherwise, a value c
 893 of -1 shall be returned, *errno* is set to indicate the error, and the file times shall not be c
 894 affected. c

895 5.6.6.4 Errors

896 If any of the following conditions occur, the *utime()* function shall return -1 and set B
 897 *errno* to the corresponding value: B

898 [EACCES] Search permission is denied by a component of the path prefix; or B
 899 the *times* argument is **NULL** and the effective user ID of the B
 900 process does not match the owner of the file and write access is B
 901 denied. B

902 [ENAMETOOLONG] c
 903 The length of the *path* argument exceeds $\{\text{PATH_MAX}\}$, or a c
 904 pathname component is longer than $\{\text{NAME_MAX}\}$ while c
 905 $\{_POSIX_NO_TRUNC\}$ is in effect. c

906 [ENOENT] The named file does not exist or the *path* argument points to an
 907 empty string.

908 [ENOTDIR] A component of the path prefix is not a directory.

909 [EPERM] The *times* argument is not **NULL** and the calling process's c
 910 effective user ID has write access but does not match the owner of c
 911 the file (if $\{_POSIX_UTIME_OWNER\}$ is in effect) and the calling B
 912 process does not have the appropriate privileges. B

913 [EROFS] The file resides on a read-only file system.

914 5.6.6.5 References

915 `<sys/stat.h>` §5.6.1. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

916 5.7 Configurable Pathname Variables B917 5.7.1 Get Configurable Pathname Variables B918 Functions: *pathconf()*, *fpathconf()* B919 5.7.1.1 Synopsis B920 `long pathconf (path, name)` C921 `char *path;` B922 `int name;` B923 `long fpathconf (fildes, name)` C924 `int fildes, name;` B925 5.7.1.2 Description B

926 The *pathconf()* and *fpathconf()* functions provide a method for the application to B
 927 determine the current value of a configurable limit or option (*variable*) that is associated B
 928 with a file or directory. B

929 For *pathconf()*, the *path* argument points to the pathname of a file or directory. For B
 930 *fpathconf()*, the *fildes* argument is an open file descriptor. B

931 The *name* argument represents the variable to be queried relative to that file or directory. B932 The following table lists the pathname variables from <limits.h> §2.9 or <unistd.h> B933 §2.10 that can be gotten by *pathconf()* or *fpathconf()*, and the symbolic constants, B934 defined in <unistd.h>, that are the corresponding values used for *name*: B

Variable	<i>name</i> Value	
FCHR_MAX	_PC_FCHR_MAX	B
LINK_MAX	_PC_LINK_MAX	B
MAX_CANON	_PC_MAX_CANON	B
MAX_INPUT	_PC_MAX_INPUT	B
NAME_MAX	_PC_NAME_MAX	B
PATH_MAX	_PC_PATH_MAX	C
PIPE_BUF	_PC_PIPE_BUF	C
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	C
_POSIX_CHOWN_SUP_GRP	_PC_CHOWN_SUP_GRP	C
_POSIX_DIR_DOTS	_PC_DIR_DOTS	C
_POSIX_GROUP_PARENT	_PC_GROUP_PARENT	C
_POSIX_LINK_DIR	_PC_LINK_DIR	C
_POSIX_NO_TRUNC	_PC_NO_TRUNC	C
_POSIX_UTIME_OWNER	_PC_UTIME_OWNER	C
_POSIX_V_DISABLE	_PC_V_DISABLE	C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

950 **5.7.1.3 Returns**

951 If the variable corresponding to *name* is not defined on the system, or if *name* is an
952 invalid value, or if the implementation does not support the association of *name* with the
953 file specified by *path*, or if the process did not have the appropriate privileges to query
954 the file specified by *path*, or *path* does not exist, the *pathconf()* function returns -1 . B
B
B

955 If the variable corresponding to *name* is not defined on the system, or if *name* is an
956 invalid value, or if the implementation does not support the association of *name* with the
957 file specified by *files*, the *fpathconf()* function returns -1 . B
B

958 Otherwise, the *pathconf()* or *fpathconf()* functions return the current variable value for
959 the file or directory. The value returned shall not be more restrictive than the
960 corresponding value described to the application when it was compiled with the
961 implementation's `<limits.h>` §2.9 or `<unistd.h>` §2.10. B
B

962 C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

Main body of the document containing several paragraphs of text. The text is extremely faint and illegible due to low contrast and blurring. It appears to be a standard prose or report format.

6. Input and Output Primitives

1 The functions in this chapter deal with input and output from files and pipes. Functions c
2 are also specified which deal with the coordination and management of file descriptors c
3 and I/O activity. c

4 6.1 Pipes

5 6.1.1 Create an Inter-Process Channel

6 Function: *pipe()*

7 6.1.1.1 Synopsis

8 int *pipe* (*fdes*)
9 int *fdes*[2];

10 6.1.1.2 Description

11 The *pipe()* function shall create a pipe and place two file descriptors, one each into the c
12 arguments *fdes*[0] and *fdes*[1], that refer to the open file descriptions for the read and c
13 write end of the pipe. Their integer values shall be the two lowest available at the time of
14 the *pipe()* function call. The O_NONBLOCK flag shall be clear on both file descriptors.
15 (The *fcntl()* function can be used to set the O_NONBLOCK flag.)

16 Data can be written to file descriptor *fdes*[1] and read from file descriptor *fdes*[0]. A
17 read on file descriptor *fdes*[0] shall access the data written to file descriptor *fdes*[1] on
18 a first-in-first-out basis.

19 8

20 An attempt to write on *fdes*[0] or to read on *fdes*[1] shall fail.

21 Upon successful completion, the *pipe()* function shall mark for update the *st_atime*, c
22 *st_ctime*, and *st_mtime* fields of the pipe. c

23 6.1.1.3 Returns

24 Upon successful completion, the function shall return a value of zero. Otherwise, a value
25 of -1 shall be returned and *errno* shall be set to indicate the error.

26 6.1.1.4 Errors

27 If any of the following conditions occur, the *pipe()* function shall return -1 and set *errno* B
28 to the corresponding value: B

29 [EMFILE] More than {OPEN_MAX} minus two file descriptors are already in
30 use by this process.

31 [ENFILE] The number of simultaneously open files in the system would B
32 exceed a system-imposed limit. B

33 B

34 6.1.1.5 References

35 *fcntl()* §6.5.2, *open()* §5.3.1, *read()* §6.4.1, *write()* §6.4.2.

36 6.2 File Descriptor Manipulation

37 6.2.1 Duplicate an Open File Descriptor

38 Functions: *dup()*, *dup2()*

39 6.2.1.1 Synopsis

40 `int dup (fildes)`

41 `int fildes;`

42 `int dup2 (fildes, fildes2)`

43 `int fildes, fildes2;`

44 6.2.1.2 Description

45 The *dup()* and *dup2()* functions provide an alternate interface to the service provided by
46 the *fcntl()* function using the *F_DUPFD* command. The call

47 `fid = dup (fildes);`

48 shall be equivalent to

49 `fid = fcntl (fildes, F_DUPFD, 0);`

50 The call

51 `fid = dup2 (fildes, fildes2);`

52 shall be equivalent to

53 `close (fildes2);`

54 `fid = fcntl (fildes, F_DUPFD, fildes2);`

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

55 except for the following:

56 If *fildest* is not a valid file descriptor, the *dup2()* function shall return [EBADF]. c

57 If *fildest* is a valid file descriptor and is equal to *fildest2*, the *dup2()* function shall
58 return *fildest2* without closing it.

59 c

60 6.2.1.3 Returns

61 Upon successful completion, the function shall return a file descriptor. Otherwise, a
62 value of -1 shall be returned and *errno* shall be set to indicate the error.

63 6.2.1.4 Errors

64 If any of the following conditions occur, the *dup()* and *dup2()* functions shall return -1 B
65 and set *errno* to the corresponding value: B

66 [EBADF] The argument *fildest* is not a valid file descriptor or *fildest2* is out of
67 range.

68 [EMFILE] The number of file descriptors would exceed {OPEN_MAX}. B

69 6.2.1.5 References

70 *close()* §6.3.1, *creat()* §5.3.2, *exec* §3.1.2, *fcntl()* §6.5.2, *open()* §5.3.1, *pipe()* §6.1.1. B

71 6.3 File Descriptor Deassignment

72 6.3.1 Close a File

73 Function: *close()*

74 6.3.1.1 Synopsis

75 `int close (fildest)`

76 `int fildest;`

77 6.3.1.2 Description

78 The *fildest* argument is a file descriptor. The *close()* function shall deallocate (i.e., make c
79 available for return by subsequent *open()*'s, etc., executed by the process) the file
80 descriptor indicated by *fildest*. All outstanding record locks owned by the process on the c
81 file descriptor indicated by *fildest* shall be removed (that is, unlocked).

82 If the *close()* function is interrupted by a signal that is to be caught, it shall return -1 B
83 with *errno* set to [EINTR] and the state of *fildest* is implementation defined. When all file B
84 descriptors associated with a pipe or FIFO special file have been closed, any data B
85 remaining in the pipe or FIFO shall be discarded. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

86 **6.3.1.3 Returns**

87 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1
88 shall be returned and *errno* shall be set to indicate the error.

89 **6.3.1.4 Errors**

90 If any of the following conditions occur, the *close()* function shall return -1 and set **B**
91 *errno* to the corresponding value: **B**

92 [EBADF] The *fildev* argument is not a valid file descriptor.

93 [EINTR] The *close* function was terminated prematurely by a signal. **B**

94 **6.3.1.5 References**

95 *creat()* §5.3.2, *dup()* §6.2.1, *exec* §3.1.2, *fcntl()* §6.5.2, *fork()* §3.1.1, *open()* §5.3.1, **B**
96 *pipe()* §6.1.1.

97 **6.4 Input and Output**98 **6.4.1 Read from a File**

99 Function: *read()*

100 **6.4.1.1 Synopsis**

```
101     int read (fildev, buf, nbyte)
102     int fildev;
103     char *buf;
104     unsigned nbyte;
```

9

105 **6.4.1.2 Description**

106 The *fildev* argument is an open file descriptor.

107 The *read()* function shall attempt to read *nbyte* bytes from the file associated with *fildev*
108 into the buffer pointed to by *buf*.

109 On a regular file or other file capable of seeking, *read()* shall start at a position in the file **c**
110 given by the file offset associated with *fildev*. Before successful return from *read()*, the **c**
111 file offset shall be incremented by the number of bytes actually read.

112 On a file not capable of seeking, the *read()* shall start from the current position. The **c**
113 value of a file offset associated with such a file is undefined. **c**

114 Upon successful completion, the *read()* function shall return the number of bytes **B**
115 actually read and placed in the buffer. This number shall never be greater than *nbyte*. **B**
116 The value returned may be less than *nbyte* if the number of bytes left in the file is less **B**
117 than *nbyte*, if the *read()* request was interrupted by a signal, or if the file is a pipe (or **B**
118 FIFO) or special file and has fewer than *nbyte* bytes immediately available for reading. **B**
119 For example, a *read()* from a file associated with a terminal may return one typed line of **B**
120 data. **B**

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 121 If a *read()* is interrupted by a signal before it reads any data, it shall return -1 with *errno* B
 122 set to [EINTR]. B
- 123 If a *read()* is interrupted by a signal after it has successfully read some data, either it B
 124 shall return -1 with *errno* set to [EINTR], or it shall return the number of bytes read. A B
 125 *read()* from a pipe or FIFO shall never return with *errno* set to [EINTR] if it has B
 126 transferred any data. B
- 127 If an end-of-file has been reached, zero shall be returned. The result of subsequent B
 128 *read()* requests on *fildev* is implementation defined. B
- 129 The value of *nbyte* shall not be greater than {INT_MAX}; otherwise, the result is A
 130 implementation defined. A
- 131 When attempting to read from an empty pipe (or FIFO):
- 132 If no process has the pipe open for writing, *read()* shall return zero to indicate B
 133 end-of-file. B
- 134 If some process has the pipe open for writing and O_NONBLOCK is set, *read()* B
 135 shall return a -1 and set *errno* [EAGAIN].
- 136 If some process has the pipe open for writing and O_NONBLOCK is clear, *read()* B
 137 shall block until some data is written or the pipe is closed by all processes that B
 138 had opened the pipe for writing. B
- 139 When attempting to read a file (other than a pipe or FIFO) that supports nonblocking A
 140 reads and has no data currently available:
- 141 If O_NONBLOCK is set, *read()* shall return a -1 and set *errno* to [EAGAIN].
- 142 If O_NONBLOCK is clear, *read()* shall block until some data becomes available. B
- 143 The use of the O_NONBLOCK flag has no effect if there is some data available. B
 144 B
- 145 For any portion of a regular file, prior to the end-of-file, that has not been written, *read()* A
 146 shall return bytes with value zero. A
- 147 Upon successful completion, the *read()* function shall mark for update the *st_atime* field C
 148 of the file. C
- 149 **6.4.1.3 Returns**
- 150 Upon successful completion, *read()* shall return an integer indicating the number of B
 151 bytes actually read. Otherwise, *read()* shall return a value of -1 and set *errno* to indicate B
 152 the error, and the content of the buffer pointed to by *buf* is indeterminate. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

153 6.4.1.4 Errors

154 If any of the following conditions occur, the *read()* function shall return -1 and set *errno* B
 155 to the corresponding value: B

156 [EAGAIN] The *O_NONBLOCK* flag is set for the file descriptor and the
 157 process would be delayed in the read operation.

158 [EBADF] The *fildev* argument is not a valid file descriptor open for reading.

159 [EINTR] The read operation was terminated due to the receipt of a signal, B
 160 and either no data was transferred or the implementation does not B
 161 report partial transfer for this file. B

162 B

163 6.4.1.5 References

164 *creat()* §5.3.2, *dup()* §6.2.1, *fcntl()* §6.5.2, *lseek()* §6.5.3, *open()* §5.3.1, *pipe()* §6.1.1, C
 165 *sigaction()* §3.3.4. C

166 6.4.2 Write to a File

167 Function: *write()*

168 6.4.2.1 Synopsis

```
169     int write (fildev, buf, nbyte)
170     int fildev;
171     char *buf;
172     unsigned nbyte;
```

9

173 6.4.2.2 Description

174 The *fildev* argument is an open file descriptor.

175 The *write()* function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf*
 176 to the file associated with the *fildev*.

177 On a regular file or other file capable of seeking, the actual writing of data shall proceed A
 178 from the position in the file indicated by the file offset associated with *fildev*. Before C
 179 successful return from *write()*, the file offset shall be incremented by the number of bytes C
 180 actually written. C

181 On a file not capable of seeking, the *write()* shall start from the current position. The C
 182 value of a file offset associated with such a file is undefined. C

183 If the *O_APPEND* flag of the file status flags is set, the file offset shall be set to the end of C
 184 the file prior to each write. C

185 If a *write()* requests that more bytes be written than there is room for (for example, the
 186 physical end of a medium), only as many bytes as there is room for shall be written. For
 187 example, suppose there is space for 20 bytes more in a file before reaching a limit. A
 188 write of 512 bytes would return 20. The next write of a non-zero number of bytes would

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

189	give a failure return (except as noted below).	
190	Upon successful completion, the <i>write()</i> function shall return the number of bytes	B
191	actually written to the file associated with <i>files</i> . This number shall never be greater than	B
192	<i>nbyte</i> .	B
193	If a <i>write()</i> is interrupted by a signal before it writes any data, it shall return -1 with	B
194	<i>errno</i> set to [EINTR].	B
195	If <i>write()</i> is interrupted by a signal after it successfully writes some data, either it shall	B
196	return -1 with <i>errno</i> set to [EINTR], or it shall return the number of bytes written. A	B
197	<i>write()</i> to a pipe or FIFO shall never return with <i>errno</i> set to [EINTR] if it has transferred	B
198	any data and <i>nbyte</i> is less than or equal to {PIPE_BUF}.	B
199	The value of <i>nbyte</i> shall not be greater than {INT_MAX}; otherwise, the result is	A
200	implementation defined.	A
201	Write requests to a pipe (or FIFO) shall be handled the same as a regular file with the	
202	following exceptions:	
203	There is no file offset associated with a pipe, hence each write request shall	C
204	append to the end of the pipe.	
205		C
206	Write requests of {PIPE_BUF} bytes or less shall not be interleaved with data	
207	from other processes doing writes on the same pipe. Writes of greater than	
208	{PIPE_BUF} bytes may have data interleaved, on arbitrary boundaries, with	B
209	writes by other processes, whether or not the O_NONBLOCK flag of the file status	B
210	flags is set.	B
211	If the O_NONBLOCK flag is clear, a write request may cause the process to block,	B
212	but on normal completion it shall return <i>nbyte</i> .	B
213	If the O_NONBLOCK flag is set, <i>write()</i> requests shall be handled differently, in	C
214	the following ways: the <i>write()</i> function shall not block the process; write	C
215	requests for {PIPE_BUF} or fewer bytes shall either succeed completely and	C
216	return <i>nbyte</i> , or return -1 and set <i>errno</i> to [EAGAIN].	C
217	When attempting to write to a file descriptor (other than a pipe or FIFO) that supports	A
218	nonblocking writes and cannot accept the data immediately:	A
219	If the O_NONBLOCK flag is clear, <i>write()</i> shall block until the data can be	B
220	accepted.	B
221	If the O_NONBLOCK flag is set, <i>write()</i> shall not block the process. If some data	B
222	can be written without blocking the process, <i>write()</i> shall write what it can and	B
223	return the number of bytes written. Otherwise, it shall return -1 and <i>errno</i> shall	B
224	be set to [EAGAIN].	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

225			9
226	Upon successful completion, the <i>write()</i> function shall mark for update the <i>st_ctime</i> and		C
227	<i>st_mtime</i> fields of the file.		C
228	6.4.2.3 Returns		
229	Upon successful completion, <i>write()</i> shall return an integer indicating the number of		8
230	bytes actually written. Otherwise, it shall return a value of <i>-1</i> and set <i>errno</i> to indicate		
231	the error.		
232	6.4.2.4 Errors		
233	If any of the following conditions occur, the <i>write()</i> function shall return <i>-1</i> and set		B
234	<i>errno</i> to the corresponding value:		B
235	[EAGAIN] The <i>O_NONBLOCK</i> flag is set for the file descriptor and the		8
236	process would be delayed in the write operation.		
237	[EBADF] The <i>fdes</i> argument is not a valid file descriptor open for writing.		
238	[EFBIG] An attempt was made to write a file that exceeds an		C
239	implementation defined maximum file size.		C
240	[EINTR] The write operation was terminated due to the receipt of a signal,		B
241	and either no data was transferred or the implementation does not		B
242	report partial transfers for this file.		B
243			C
244	[ENOSPC] There is no free space remaining on the device containing the file.		
245	[EPIPE] An attempt is made to write to a pipe (or FIFO) that is not open for		
246	reading by any process. A <i>SIGPIPE</i> signal shall also be sent to the		C
247	process.		C
248			B
249	6.4.2.5. References		
250	<i>creat()</i> §5.3.2, <i>dup()</i> §6.2.1, <i>fcntl()</i> §6.5.2, <i>lseek()</i> §6.5.3, <i>open()</i> §5.3.1, <i>pipe()</i> §6.1.1,		C
251	<i>sigaction()</i> §3.3.4.		C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

252 6.5 Control Operations on Files

253 6.5.1 Data Definitions for File Control Operations

254 6.5.1.1 Synopsis

255 #include <fcntl.h>

256 6.5.1.2 Description

257 The header <fcntl.h> §6.5.1 defines the following *requests* and *arguments* for the *fcntl()*
258 and *open()* functions.259 6.5.1.2.1 *cmd* values for *fcntl()*

<u>Constant</u>	<u>Description</u>	
F_DUPFD	Duplicate file descriptor	B
F_GETFD	Get file descriptor flags	B
F_GETLK	Get record locking information	B
F_SETFD	Set file descriptor flags	B
F_GETFL	Get file status flags	B
F_SETFL	Set file status flags	B
F_SETLK	Set record locking information	B
F_SETLKW	Set record locking information; wait if blocked	B

269 6.5.1.2.2 File descriptor flags used for *fcntl()*

<u>Constant</u>	<u>Description</u>	
FD_CLOEXEC	Close the file descriptor upon execution of an <i>exec</i> function	B

272 6.5.1.2.3 *l_type* values for record locking with *fcntl()*

<u>Constant</u>	<u>Description</u>	
F_RDLCK	Shared or read lock	B
F_UNLCK	Unlock	B
F_WRLCK	Exclusive or write lock	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

277 6.5.1.2.4 *oflag* values for *open()*

<u>Constant</u>	<u>Description</u>
O_CREAT	Create file if it doesn't exist
O_EXCL	Exclusive use flag
O_TRUNC	Truncate flag

282 6.5.1.2.5 File status flags used for *open()* and *fcntl()*

<u>Constant</u>	<u>Description</u>
O_APPEND	Set append mode
O_NONBLOCK	No delay

286 6.5.1.2.6 File access modes used for *open()* and *fcntl()*

<u>Constant</u>	<u>Description</u>
O_RDONLY	Open for reading only
O_RDWR	Open for reading and writing
O_WRONLY	Open for writing only

291 6.5.1.2.7 Mask for use with file access modes

<u>Constant</u>	<u>Description</u>
O_ACCMODE	Mask for file access modes

294 6.5.1.3 References

295 *fcntl()* §6.5.2, *open()* §5.3.1.

296 6.5.2 File Control

297 Function: *fcntl()*

298 6.5.2.1 Synopsis

```

299     #include <sys/types.h>
300     #include <unistd.h>
301     #include <fcntl.h>
302     int fcntl (fdes, cmd, ...)
303     int fdes, cmd;

```

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

304 6.5.2.2 Description

305 The function *fcntl()* provides for control over open files. The argument *fil* is a file c
 306 descriptor.

307 The available values for *cmd* are defined in the header `<fcntl.h>` §6.5.1, which shall
 308 include:

309 **F_DUPFD** Return a new file descriptor which is the lowest numbered
 310 available (i.e., not already open) file descriptor greater than or c
 311 equal to the third argument, *arg*, taken as an integer of type *int*. c
 312 The new file descriptor refers to the same open file description as c
 313 the original file descriptor, and shares any locks. c

314 The **FD_CLOEXEC** flag associated with the new file descriptor is c
 315 cleared to keep the file open across calls to the *exec* family of c
 316 functions.

317 **F_GETFD** Get the file descriptor flags defined in Table 6.5.1.2.2 that are s
 318 associated with the file descriptor *fil*. If the **FD_CLOEXEC** bit c
 319 in the third argument, taken as type *int*, is zero the file shall remain
 320 open across *exec* functions; otherwise the file shall be closed upon c
 321 successful execution of the *exec* function. File descriptor flags are c
 322 associated with a single file descriptor and do not affect other file c
 323 descriptors that refer to the same file. c

324 **F_SETFD** Set the file descriptor flags defined in Table 6.5.1.2.2, that are s
 325 associated with *fil*, to the third argument, *arg*, taken as type *int*.
 326 This is zero or **FD_CLOEXEC**, as described for **F_GETFD**. c

327 **F_GETFL** Get the file status flags, defined in Table 6.5.1.2.5, and file access B
 328 modes for the open file description associated with *fil*. The file B
 329 access modes defined in Table 6.5.1.2.6 can be extracted from the B
 330 return value using the mask **O_ACCMODE**, which is defined in B
 331 `<fcntl.h>` §6.5.1. File status flags and file access modes are c
 332 associated with the open file description and do not affect other file c
 333 descriptors that refer to the same file with different open file c
 334 descriptions. c

335 **F_SETFL** Set the file status flags, defined in Table 6.5.1.2.5, for the open file c
 336 description associated with *fil* from the corresponding bits in
 337 the third argument, *arg*, taken as type *int*. The file access mode
 338 shall not be changed. If any other bits are set in *arg*, the result is
 339 implementation defined. c

340 The following commands are available for record locking. Record locking shall be
 341 supported for regular files, and may be supported for other files. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

342	F_GETLK	Get the first lock which blocks the lock description pointed to by	A
343		the third argument, <i>arg</i> , taken as a pointer to type <i>struct flock</i> (see	B
344		below). The information retrieved overwrites the information	A
345		passed to <i>fcntl()</i> in the structure <i>flock</i> . If no lock is found that	A
346		would prevent this lock from being created, then the structure shall	B
347		be left unchanged except for the lock type which shall be set to	A
348		F_UNLCK .	A
349	F_SETLK	Set or clear a file segment lock according to the lock description	A
350		pointed to by the third argument, <i>arg</i> , taken as a pointer to type	B
351		<i>struct flock</i> (see below). F_SETLK is used to establish shared (or	A
352		read) locks (F_RDLCK) or exclusive (or write) locks, (F_WRLCK),	A
353		as well as remove either type of lock (F_UNLCK). F_RDLCK ,	A
354		F_WRLCK , and F_UNLCK are defined by the <code><fcntl.h></code> §6.5.1	A
355		header. If a shared or exclusive lock cannot be set, <i>fcntl()</i> shall	B
356		return immediately.	A
357	F_SETLKW	This command is the same as F_SETLK except that if a shared or	A
358		exclusive lock is blocked by other locks, the process shall wait	B
359		until the request can be satisfied. If a signal that is to be caught is	B
360		received while <i>fcntl()</i> is waiting for a region, the <i>fcntl()</i> shall be	B
361		interrupted. Upon return from the process's signal handler, <i>fcntl()</i>	B
362		shall return <code>-1</code> with <i>errno</i> set to <code>[EINTR]</code> , and the lock operation	B
363		shall not be done.	B
364		The structure <i>flock</i> , defined by the <code><fcntl.h></code> §6.5.1 header, describes a lock. It describes	A
365		the type (<i>l_type</i>), starting offset (<i>l_whence</i>), relative offset (<i>l_start</i>), size (<i>l_len</i>), and	A
366		process-ID (<i>l_pid</i>):	A

Member Type	Member Name	Description	
<i>short</i>	<i>l_type</i>	F_RDLCK , F_WRLCK , F_UNLCK	A
<i>short</i>	<i>l_whence</i>	flag for starting offset	A
<i>off_t</i>	<i>l_start</i>	relative offset in bytes	B
<i>off_t</i>	<i>l_len</i>	size; if 0 then until EOF	B
<i>int</i>	<i>l_pid</i>	process ID of the process holding the lock, returned with F_GETLK	C B

375	When a shared lock has been set on a segment of a file, other processes shall be able to	A
376	set shared locks on that segment or a portion of it. A shared lock prevents any other	A
377	process from setting an exclusive lock on any portion of the protected area. A request for	B
378	a shared lock shall fail if the file descriptor was not opened with read access.	B
379	An exclusive lock shall prevent any other process from setting a shared lock or an	A
380	exclusive lock on any portion of the protected area. A request for an exclusive lock shall	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

- 381 fail if the file descriptor was not opened with write access. B
- 382 The value of *l_whyence* is SEEK_SET, SEEK_CUR, or SEEK_END to indicate that the A
 383 relative offset, *l_start* bytes, will be measured from the start of the file, current position, A
 384 or end of the file, respectively. The value of *l_len* is the number of consecutive bytes to A
 385 be locked. If *l_len* is negative, the result is implementation defined. The *l_pid* field is B
 386 only used with F_GETLK to return the process ID of the process holding a blocking lock. B
- 387 Locks may start and extend beyond the current end of a file, but shall not start or extend C
 388 before the beginning of the file. A lock shall be set to extend to the end of file if *l_len* is C
 389 set to zero. If the *flock struct* has *l_whyence* and *l_start* that point to the beginning of the B
 390 file, and *l_len* of zero, the entire file shall be locked. B
- 391 The calling process shall have only one type of lock set for each byte in the file. Before B
 392 successful return from a F_SETLK or F_SETLKW request, the previous lock type for each B
 393 byte in the specified region shall be replaced by the new lock type. All locks associated A
 394 with a file for a given process shall be removed when a file descriptor for that file is A
 395 closed by that process or the process holding that file descriptor terminates. Locks are B
 396 not inherited by a child process created using the *fork()* function. A
- 397 A potential for deadlock occurs if a process controlling a locked region is put to sleep by B
 398 attempting to lock another process's locked region. If the system detects that sleeping A
 399 until a locked region is unlocked would cause a deadlock, the *fcntl()* function shall fail A
 400 with an [EDEADLK] error.

401 6.5.2.3 Returns

- 402 Upon successful completion, the value returned shall depend on *cmd* as follows: C

Request	Return Value	
F_DUPFD	A new file descriptor.	
F_GETFD	Value of the flags defined in Table 6.5.1.2.2, but the return value shall not be negative.	B
F_SETFD	Value other than -1.	
F_GETFL	Value of file status flags and access modes, but the return value shall not be negative.	C C
F_SETFL	Value other than -1.	
F_GETLK	Value other than -1.	A
F_SETLK	Value other than -1.	A
F_SETLKW	Value other than -1.	A

- 411 Otherwise, a value of -1 shall be returned and *errno* shall be set to indicate the error. C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

412 6.5.2.4 Errors

413		If any of the following conditions occur, the <i>fcntl()</i> function shall return -1 and set <i>errno</i>	B
414		to the corresponding value:	B
415	[EACCES]	The argument <i>cmd</i> is <i>F_SETLK</i> , the type of lock (<i>l_type</i>) is a	A
416		shared lock (<i>F_RDLCK</i>) or exclusive lock (<i>F_WRLCK</i>), and the	A
417		segment of a file to be locked is already exclusive-locked by	A
418		another process, or the type is an exclusive lock and some portion	C
419		of the segment of a file to be locked is already shared-locked or	A
420		exclusive-locked by another process.	
421	[EBADF]	The <i>fildev</i> argument is not a valid file descriptor.	
422		The argument <i>cmd</i> is <i>F_SETLK</i> or <i>F_SETLKW</i> , the type of lock	A
423		(<i>l_type</i>) is a shared lock (<i>F_RDLCK</i>), and <i>fildev</i> is not a valid file	A
424		descriptor open for reading.	A
425		The argument <i>cmd</i> is <i>F_SETLK</i> or <i>F_SETLKW</i> , the type of lock	A
426		(<i>l_type</i>) is an exclusive lock (<i>F_WRLCK</i>), and <i>fildev</i> is not a valid	A
427		file descriptor open for writing.	A
428	[EINTR]	The argument <i>cmd</i> is <i>F_SETLKW</i> and the function was interrupted	B
429		by a signal.	B
430	[EINVAL]	The argument <i>cmd</i> is <i>F_DUPFD</i> and the third argument is negative	C
431		or greater than or equal to $\{OPEN_MAX\}$.	
432		The argument <i>cmd</i> is <i>F_GETLK</i> , <i>F_SETLK</i> , or <i>F_SETLKW</i> and the	A
433		data <i>arg</i> points to is not valid, or <i>fildev</i> refers to a file that does not	B
434		support locking.	B
435	[EMFILE]	The argument <i>cmd</i> is <i>F_DUPFD</i> and $\{OPEN_MAX\}$ file descriptors	
436		are currently in use by this process.	A
437	[ENOLCK]	The argument <i>cmd</i> is <i>F_SETLK</i> or <i>F_SETLKW</i> and satisfying the	B
438		lock or unlock request would result in the number of locked	B
439		regions in the system exceeding a system-imposed limit.	B
440		For each of the following conditions, if the condition is detected, the <i>fcntl()</i> function	B
441		shall return -1 and set <i>errno</i> to the corresponding value:	B
442	[EDEADLK]	The argument <i>cmd</i> is <i>F_SETLKW</i> and a deadlock condition was	B
443		detected.	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

444 6.5.2.5 References

445 *close()* §6.3.1, *exec* §3.1.2, *open()* §5.3.1, *<fcntl.h>* §6.5.1, *sigaction()* §3.3.4. c

446 6.5.3 Reposition Read/Write File Offset c

447 Function: *lseek()*

448 6.5.3.1 Synopsis

449 `#include <sys/types.h>`

450 `#include <unistd.h>`

451 `off_t lseek (fildes, offset, whence)`

452 `int fildes, whence;` c

453 `off_t offset;`

454 6.5.3.2 Description

455 The *fildes* argument is an open file descriptor. The *lseek()* function shall set the file c
456 offset for the open file description associated with *fildes* as follows: c

457 If *whence* is `SEEK_SET`, the offset is set to *offset* bytes. c

458 If *whence* is `SEEK_CUR`, the offset is set to its current value plus *offset* bytes. c

459 If *whence* is `SEEK_END`, the offset is set to the size of the file plus *offset* bytes. c

460 The symbolic constants `SEEK_SET`, `SEEK_CUR`, `SEEK_END` are defined in the header
461 *<unistd.h>* §2.10.

462 Some devices are incapable of seeking. The value of the file offset associated with such c
463 a device is undefined. c

464 The *lseek()* function shall allow the file offset to be set beyond the end of existing data in c
465 the file. If data is later written at this point, subsequent reads of data in the gap shall A
466 return bytes with the value zero until data is actually written into the gap.

467 The *lseek()* function shall not, by itself, extend the size of a file. 9

468 6.5.3.3 Returns

469 Upon successful completion, the function shall return the resulting offset location as c
470 measured in bytes from the beginning of the file. Otherwise, it shall return a value of

471 `(off_t) -1`, shall set *errno* to indicate the error, and the file offset shall remain unchanged. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

472 6.5.3.4 Errors

473 If any of the following conditions occur, the *lseek()* function shall return -1 and set B
 474 *errno* to the corresponding value: B

475 [EBADF] The *fdes* argument is not a valid file descriptor.

476 [EINVAL] The *whence* argument is not a proper value, or the resulting file c
 477 offset would be invalid. c

478 [ESPIPE] The *fdes* argument is associated with a pipe or FIFO.

479 6.5.3.5 References

480 *creat()* §5.3.2, *dup()* §6.2.1, *fcntl()* §6.5.2, *open()* §5.3.1, *read()* §6.4.1, *sigaction()* c

481 §3.3.4, *write()* §6.4.2, <unistd.h> §2.10. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

7. Device- and Class-Specific Functions

1 7.1 General Terminal Interface 8

2 7.1.1 Interface Characteristics 8

3 7.1.1.1 Description 8

4 This section describes a general terminal interface that shall be provided to control A
5 asynchronous communications ports. It is implementation defined whether this interface A
6 supports network connections and/or synchronous ports. 8

7 7.1.1.2 Opening a Terminal Device File 8

8 When a terminal file is opened, it normally causes the process to wait until a connection 8
9 is established. In practice, user programs seldom open these files; they are opened by 8
10 special programs and become a user's standard input, output, and error files. 8

11 As described in *open()* §5.3.1, opening a terminal device file with the `O_NONBLOCK` 8
12 flag clear shall cause the process to block until the connection is established. If the 8
13 `O_NONBLOCK` flag is set, the *open()* function shall return a file descriptor without 8
14 waiting for a connection to be established. 8

15 7.1.1.3 Process Groups 8

16 A terminal may have a distinguished process group associated with it. This distinguished 8
17 process group plays a special role in handling signal-generating input characters, as 8
18 discussed below in Special Characters §7.1.1.10. 8

19 If the implementation supports the Job Control Option (if `{_POSIX_JOB_CONTROL}` is c
20 defined; see Symbolic Constants §2.10), command interpreter processes* supporting job c
21 control can allocate the terminal to different *jobs*, or process groups, by placing related 8
22 processes in a single process group and associating this process group with the terminal. 8
23 A terminal's associated process group may be set or examined by a process, assuming the c

* The P1003.2 Working Group is working on a definition and description of command interpreters. See Shell and Utilities §A.2.2.

24 permission requirements in this section are met; see *tcgetpgrp()* §7.2.3 and *tcsetpgrp()* C
 25 §7.2.4. The terminal interface aids in this allocation by restricting access to the terminal 8
 26 by processes that are not in the current process group; see Job Access Control §7.1.1.5. 8

27 **7.1.1.4 The Controlling Terminal** 8

28 A terminal may belong to a process as its *controlling terminal*. If a process that is a 8
 29 “session process group leader,” and that does not have a controlling terminal, opens a 8
 30 terminal file not already associated with a process group, the terminal associated with 8
 31 that terminal file becomes the controlling terminal for that process, and the terminal’s 8
 32 distinguished process group is set to the process group of that process. C

33 The controlling terminal is inherited by a child process during a *fork()* function. A 8
 34 process relinquishes its controlling terminal when it changes its process group using a 8
 35 *setpgrp()* function. 8

36 When controlling process terminates, the distinguished process group of its controlling C
 37 terminal is set to zero (indicating no distinguished process group). This allows the 8
 38 terminal to be acquired as a controlling terminal by a new session process group leader. 8

39 **7.1.1.5 Job Access Control** 8

40 If a process is in the distinguished process group of its controlling terminal, or the C
 41 distinguished process group is zero (that is, if the process is a *foreground process*), then C
 42 read operations shall be allowed as described below in Input Processing and Reading C
 43 Characters §7.1.1.6. For those implementations that do not support the Job Control C
 44 Option, a background process shall also be allowed to read from its controlling terminal. C
 45 For those implementations that support the Job Control Option, if a process is not in the C
 46 (non-zero) distinguished process group of its controlling terminal (that is, if the process is C
 47 a *background process*), then any attempts to read from that terminal shall cause the C
 48 process group to be sent a SIGTTIN signal unless the reading process is ignoring or C
 49 blocking the SIGTTIN signal. If the process is ignoring or blocking the SIGTTIN signal, C
 50 the process is instead returned an [EIO] error and no signal is sent to the process. The C
 51 default action of the SIGTTIN signal is to stop the process to which it is sent. See Signal C
 52 Names §3.3.1. C

53 It is frequently undesirable for background processes to write to their controlling A
 54 terminal. If TOSTOP (see Local Modes §7.1.2.6) is set, then attempts by a background A
 55 process to write to its controlling terminal shall cause the process group to be sent a A
 56 SIGTTOU signal, which, by default, will cause the members of the process group to stop. A
 57 If TOSTOP is not set or the process is ignoring or blocking SIGTTOU signals, the process A
 58 is allowed to write to the terminal and the SIGTTOU signal is not sent. Certain calls that A
 59 set terminal parameters are treated in this same fashion, except that TOSTOP is ignored; A
 60 however, the effect is identical to that of terminal writes when TOSTOP is set. See C
 61 Control Functions §7.2. C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

62 7.1.1.6 Input Processing and Reading Characters 8

63 A terminal device associated with a terminal device file may operate in full-duplex mode, 9
 64 so that characters may arrive even while output is occurring. Each terminal device file 9
 65 has associated with it an *input queue*, into which incoming characters are stored by the 9
 66 system before being read by a process. The system may impose a limit, {MAX_INPUT}, C
 67 on the number of bytes that may be stored in the input queue. The behavior of the A
 68 system when this limit is exceeded is implementation defined. A

69 Two general kinds of input processing are available, determined by whether the terminal 8
 70 device file is in canonical mode or non-canonical mode. These modes are described in 8
 71 the Canonical Mode Input Processing §7.1.1.7 and Non-Canonical Mode Input 8
 72 Processing §7.1.1.8. Additionally, input characters are processed according to the 8
 73 *c_iflag* (see Input Modes §7.1.2.3) and *c_lflag* (see Local Modes §7.1.2.6) fields. Such 8
 74 processing can include *echoing*; which in general means transmitting input characters 8
 75 immediately back to the terminal when they are received from the terminal. This is 8
 76 useful for terminals that can operate in full-duplex mode. The manner in which 8
 77 characters are provided to a process reading from a terminal device file is very dependent 8
 78 on whether the terminal file is in canonical or non-canonical mode. 8

79 Another dependency is whether the O_NONBLOCK flag is set by *open()* or *fcntl()*. If the 8
 80 O_NONBLOCK flag is clear, then the read request shall block until data is available B
 81 or a signal has been received. If the O_NONBLOCK flag is set, then the read request shall 8
 82 complete, without blocking, in one of three ways: 8

- 83 1. If there is enough data available to satisfy the entire request, the read shall 8
 84 complete successfully, having read all the requested data, and return the A
 85 number of bytes read. A
- 86 2. If there is not enough data available to satisfy the entire request, the read 8
 87 shall complete successfully, having read as much data as possible, and 8
 88 return the number of bytes it was able to read. 8
- 89 3. If there is no data available, the read shall return a -1, with *errno* set to 8
 90 EAGAIN. 8

91 When data is available depends on whether the input processing mode is canonical or C
 92 non-canonical. The following sections, Canonical Mode Input Processing §7.1.1.7 and C
 93 Non-Canonical Mode Input Processing §7.1.1.8, describe each of these input C
 94 processing modes. C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

95	7.1.1.7 Canonical Mode Input Processing	8
96	In canonical mode input processing, terminal input is processed in units of lines. A line	8
97	is delimited by a new-line ('\n') character, an end-of-file (EOF) character, or an end-	9
98	of-line (EOL) character. See the Special Characters §7.1.1.10 for more information on	9
99	EOF and EOL. This means that a read request shall not be satisfied until an entire line has	C
100	been typed, or a signal has been received. Also, no matter how many characters are	8
101	requested in the read call, at most one line shall be returned. It is not, however,	8
102	necessary to read a whole line at once; any number of characters, even one, may be	8
103	requested in a read without losing information.	8
104	If {MAX_CANON} is defined, it is a limit on the number of bytes in a line. The behavior	A
105	of the system when this limit is exceeded is implementation defined.	A
106	Erase and kill processing occurs during input. The ERASE character erases the last	C
107	character typed in the current input line.	C
108	ERASE shall not erase beyond the beginning of the current input line. The KILL	8
109	character kills (deletes) the entire current input line, and optionally outputs a new-line	8
110	character. All these characters operate on a keystroke basis, independently of any	8
111	backspacing or tabbing that may have been done.	8
112		9
113	7.1.1.8 Non-Canonical Mode Input Processing	8
114	In non-canonical mode input processing, input characters are not assembled into lines,	8
115	and erase and kill processing does not occur. The values of the special characters MIN	B
116	and TIME are used to determine how to process the characters received. MIN and TIME	C
117	are defined by the <i>c_cc</i> array of special control characters.	C
118	MIN represents the minimum number of characters that should be received when the read	8
119	is satisfied (i.e., the characters are returned to the user). TIME is a timer of 0.1 second	B
120	granularity that is used to time out bursty and short term data transmissions. If MIN is	B
121	greater than {MAX_INPUT}, the response to the request is implementation defined. The	8
122	four possible values for MIN and TIME and their interactions are described below.	B
123	7.1.1.8.1 Case A: MIN > 0, TIME > 0	B
124	In this case TIME serves as an intercharacter timer and is activated after the first character	8
125	is received. Since it is an intercharacter timer, it is reset after a character is received. The	8
126	interaction between MIN and TIME is as follows: as soon as one character is received,	B
127	the intercharacter timer is started. If MIN characters are received before the	B
128	intercharacter timer expires (remember that the timer is reset upon receipt of each	8
129	character), the read is satisfied. If the timer expires before MIN characters are received,	B
130	the characters received to that point are returned to the user. Note that if TIME expires at	B
131	least one character shall be returned because the timer would not have been enabled	8
132	unless a character was received. In this case (MIN > 0, TIME > 0) the read shall block	C
133	until the MIN and TIME mechanisms are activated by the receipt of the first character.	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

134 7.1.1.8.2 Case B: $\text{MIN} > 0$, $\text{TIME} = 0$ B

135 In this case, since the value of TIME is zero, the timer plays no role and only MIN is B
 136 significant. A pending read is not satisfied until MIN characters are received (i.e., the C
 137 pending read shall block until MIN characters are received). A program that uses this 8
 138 case to read record-based terminal I/O may block indefinitely in the read operation. B

139 7.1.1.8.3 Case C: $\text{MIN} = 0$, $\text{TIME} > 0$ B

140 In this case, since $\text{MIN} = 0$, TIME no longer represents an intercharacter timer. It now 8
 141 serves as a read timer that is activated as soon as the *read()* function is processed. A 8
 142 read is satisfied as soon as a single character is received or the read timer expires. Note 8
 143 that in this case if the timer expires, no character shall be returned. If the timer does not 8
 144 expire, the only way the read can be satisfied is if a character is received. In this case the 8
 145 read shall not block indefinitely waiting for a character; if no character is received within B
 146 $\text{TIME} \times 0.1$ seconds after the read is initiated, the read shall return with zero characters. B

147 7.1.1.8.4 Case D: $\text{MIN} = 0$, $\text{TIME} = 0$ B

148 The minimum of either the number of characters requested or the number of characters C
 149 currently available shall be returned without waiting for more characters to be input. C

150 7.1.1.8.5 Comparison of the Different Cases of MIN , TIME Interaction B

151 Some points to note about MIN and TIME : B

152 1. In the preceding explanations one may notice that the interactions of MIN and B
 153 TIME are not symmetric. For example, when $\text{MIN} > 0$ and $\text{TIME} = 0$, TIME has no B
 154 effect. However, in the opposite case where $\text{MIN} = 0$ and $\text{TIME} > 0$, both MIN and B
 155 TIME play a role in that MIN is satisfied with the receipt of a single character. B

156 2. Also note that in case A ($\text{MIN} > 0$, $\text{TIME} > 0$), TIME represents an intercharacter B
 157 timer while in case C ($\text{MIN} = 0$, $\text{TIME} > 0$) TIME represents a read timer. 8

158 These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, B
 159 where $\text{MIN} > 0$, exist to handle burst mode activity (e.g., file transfer programs) where a B
 160 program would like to process at least MIN characters at a time. In case A, the 8
 161 intercharacter timer is activated by a user as a safety measure; while in case B, it is 8
 162 turned off. 8

163 Cases C and D exist to handle single character timed transfers. These cases are readily 8
 164 adaptable to screen-based applications that need to know if a character is present in the 8
 165 input queue before refreshing the screen. In case C the read is timed; while in case D, it is 8
 166 not. 8

167 Another important note is that MIN is always just a minimum. It does not denote a record B
 168 length. That is, if a program does a read of 20 bytes, MIN is 10, and 25 characters are 8
 169 present, 20 characters shall be returned to the user. 8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

170	7.1.1.9 Writing Characters and Output Processing		8
171	When a process writes one or more characters to a terminal device file, they are		8
172	processed according to the <i>c_oflag</i> (see Output Modes §7.1.2.4) The implementation		9
173	may provide a buffering mechanism; as such, when a call to <i>write()</i> completes, all of the		9
174	characters written have been scheduled for transmission to the device, but the		9
175	transmission will not necessarily have completed.		9
176			C
177	7.1.1.10 Special Characters		8
178	Certain characters have special functions on input and/or output. These functions are		8
179	summarized as follows:		8
180	INTR	Special character on input and is recognized if the ISIG flag is	A
181		enabled. Generates a SIGINT signal which is sent to all processes	8
182		in the distinguished process group associated with the terminal	8
183	QUIT	Special character on input and is recognized if the ISIG flag is	A
184		enabled. Generates a SIGQUIT signal which is sent to all processes	9
185		in the distinguished process group associated with the terminal.	9
186	ERASE	Special character on input and is recognized if the ICANON flag is	A
187		set. Erases the preceding character. It shall not erase beyond the	9
188		start of a line, as delimited by an NL, EOF, or EOL character.	A
189	KILL	Special character on input and is recognized if the ICANON flag is	A
190		set. Deletes the entire line, as delimited by a NL, EOF, or EOL	8
191		character.	8
192			9
193	EOF	Special character on input and is recognized if the ICANON flag is	A
194		set. When received, all the characters waiting to be read are	8
195		immediately passed to the program, without waiting for a new-	8
196		line, and the EOF is discarded. Thus, if there are no characters	8
197		waiting (that is, the EOF occurred at the beginning of a line), zero	8
198		characters shall be passed back, representing an end-of-file	8
199		indication.	8
200	NL	Special character on input and is recognized if the ICANON flag is	A
201		set. Is the line delimiter ('\n'). It cannot be changed.	9
202	EOL	Special character on input and is recognized if the ICANON flag is	A
203		set. Is an additional line delimiter, like NL.	8
204	SUSP	Special character on input and is recognized if the ISIG flag is	A
205		enabled (Job Control Option only). Generates a SIGTSTP signal	8
206		which is sent to all processes in the distinguished process group	8
207		associated with the terminal.	8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

208	STOP	Special character on both input and output and is recognized if the	A
209		IXON (input) or IXOFF (output) flag is set. (ASCII DC3) Can be	8
210		used to temporarily suspend output. It is useful with CRT	8
211		terminals to prevent output from disappearing before it can be	C
212		read.	C
213	START	Special character on both input and output and is recognized if the	A
214		IXON (input) or IXOFF (output) flag is set. (ASCII DC1) Can be	8
215		used to resume output that has been suspended by a STOP	C
216		character.	C
217	The START and STOP characters cannot be changed. The values for INTR, QUIT,		8
218	ERASE, KILL, EOF, EOL, SUSP (Job Control Option only), shall be changeable to suit		A
219	individual tastes.		B
220	If {_POSIX_V_DISABLE} is in effect for the terminal file, special character functions can		B
221	be disabled individually.		B
222	If two or more special characters have the same value, the function performed when that		8
223	character is received is undefined.		8
224	A special character is recognized not only by its value, but also by its context; e.g., an		C
225	implementation may define multi-byte sequences that have a meaning different from the		C
226	meaning of the bytes when considered individually. Implementations may also define		C
227	additional single-byte functions.		C
228	7.1.1.11 Modem Disconnect		8
229	When a modem disconnect is detected by the terminal interface, a SIGHUP signal is sent		8
230	to all processes in the distinguished process group associated with the terminal. Unless		8
231	other arrangements have been made, this signal causes the processes to terminate. If		8
232	SIGHUP is ignored or caught, any subsequent read returns with an end-of-file indication		C
233	until the device is closed. Thus programs that read a terminal file and test for end-of-file		8
234	can terminate appropriately after a disconnect.		8
235	7.1.1.12 Closing a Terminal Device File		8
236	The last process to close a terminal device file shall cause any output to be sent to the		B
237	device and any input to be discarded. If HUPCL is set in the control modes, and the		B
238	communications port supports a disconnect function, the terminal device shall perform a		B
239	disconnect.		B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

240	7.1.2 Settable Parameters	8
241	7.1.2.1 Synopsis	8
242	#include <termios.h>	8
243	7.1.2.2 <i>termios</i> Structure	8
244	Routines that need to control certain terminal I/O characteristics shall do so by using the	8
245	<i>termios</i> structure as defined in the header <termios.h>. The members of this structure	8
246	include (but are not limited to):	8

<u>Member Type</u>	<u>Array Size</u>	<u>Member Name</u>	<u>Description</u>	
unsigned long		c_iflag	input modes	A
unsigned long		c_oflag	output modes	A
unsigned long		c_cflag	control modes	A
unsigned long		c_lflag	local modes	A
unsigned char []	NCCS	c_cc	control chars	C

254	The total size of the <i>termios</i> structure is implementation defined.	C
255	7.1.2.3 Input Modes	8
256	The <i>c_iflag</i> field describes the basic terminal input control:	8

<u>Mask Name</u>	<u>Description</u>	
BRKINT	Signal interrupt on break.	8
ICRNL	Map CR to NL on input.	8
IGNBRK	Ignore break condition.	8
IGNCR	Ignore CR.	8
IGNPAR	Ignore characters with parity errors.	8
INLCR	Map NL to CR on input.	8
INPCK	Enable input parity check.	8
ISTRIP	Strip character.	8
IXOFF	Enable start/stop input control.	9
IXON	Enable start/stop output control.	8
PARMRK	Mark parity errors.	8

274	If IGNBRK is set, a break condition (a character framing error with data all zeroes)	8
275	detected on input is ignored, that is, not put on the input queue and therefore not read by	8
276	any process. Otherwise if BRKINT is set, the break condition shall generate a single	8
277	SIGINT signal and flush both the input and output queues. If neither IGNBRK or BRKINT	B
278	is set, a break condition is read as a single '\0'.	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

- 279 If IGNPAR is set, a byte with a framing or parity error (other than break) is ignored. C
- 280 If PARMRK is set, a byte with a framing and parity error (other than break) that is not B
 281 ignored is given to the application as the three-character sequence '\377', '\0', X, where B
 282 '\377', '\0' is a two-character flag preceding each sequence and X is the data of the B
 283 character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid B
 284 character of '\377' is given to the application as '\377', '\377'. If PARMRK is not set, B
 285 a framing or parity error (other than break) that is not ignored is given to the application B
 286 as a single character '\0'. B
- 287 If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity 8
 288 checking is disabled, allowing output parity generation without input parity errors. Note 8
 289 that whether input parity checking is enabled or disabled is independent of whether parity 8
 290 detection is enabled or disabled. If parity detection is enabled but input parity checking 8
 291 is disabled, the hardware to which the terminal is connected shall recognize the parity bit, 8
 292 but the terminal special file shall not check whether this bit is set correctly or not. 8
- 293 If ISTRIP is set, valid input characters are first stripped to 7 bits, otherwise all 8 bits are 8
 294 processed. 8
- 295 If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, 8
 296 a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR 8
 297 character is translated into a NL character. 8
- 298 If IXON is set, start/stop output control is enabled. A received STOP character shall 8
 299 suspend output and a received START character shall restart output. When IXON is set, B
 300 START and STOP characters are not read, but merely perform flow control functions. B
 301 When IXON is not set, the START and STOP characters are read. B
- 302 9
- 303 If IXOFF is set, start/stop input control is enabled. The system shall transmit STOP 9
 304 characters, which are intended to cause the terminal device to stop transmitting data, as 9
 305 needed to prevent the number of characters in the input queue from exceeding C
 306 {MAX_INPUT}, and shall transmit START characters, which are intended to cause the 8
 307 terminal device to resume transmitting data, as soon as the device can continue 8
 308 transmitting data without risk of overflowing the input queue. The precise conditions 8
 309 under which STOP and START characters are transmitted are implementation defined. 9
- 310 The initial input control value after *open()* is implementation defined. 8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

- 311 7.1.2.4 Output Modes 8
- 312 The *c_oflag* field specifies the terminal interface's treatment of output: 8

<u>Mask Name</u>	<u>Description</u>	
OPOST	Perform output processing.	8

- 318 If OPOST is set, output characters are processed in an implementation defined fashion so 9
- 319 that lines of text are modified to appear appropriately on the terminal device, otherwise 8
- 320 characters are transmitted without change. 8
- 321 The initial output control value after *open()* is implementation defined. 8
- 322 7.1.2.5 Control Modes 8
- 323 The *c_cflag* field describes the hardware control of the terminal; not all values specified 8
- 324 are required to be supported by the underlying hardware: 8

<u>Mask Name</u>	<u>Description</u>	
CLOCAL	Ignore modem status lines.	8
CREAD	Enable receiver.	8
CSIZE	Character size:	8
CS5	5 bits	8
CS6	6 bits	8
CS7	7 bits	8
CS8	8 bits	8
CSTOPB	Send two stop bits, else one.	8
HUPCL	Hang up on last close.	8
PARENB	Parity enable.	8
PARODD	Odd parity, else even.	8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

341 In addition, the input and output baud rates are also stored in the *c_cflag* field. The 9
 342 following values are supported: 9

<u>Name</u>	<u>Description</u>	9
B0	Hang up	9
B50	50 baud	9
B75	75 baud	9
B110	110 baud	9
B134	134.5 baud	9
B150	150 baud	9
B200	200 baud	9
B300	300 baud	9
B600	600 baud	9
B1200	1200 baud	9
B1800	1800 baud	9
B2400	2400 baud	9
B4800	4800 baud	9
B9600	9600 baud	9
B19200	19200 baud	9
B38400	38400 baud	9

360 The following interfaces are provided for getting and setting the values of the input and A
 361 output baud rates: A

362 `int cfgetospeed (termios_p)` C
 363 `struct termios *termios_p;` A

364 `int cfsetospeed (termios_p, speed)` C
 365 `struct termios *termios_p;` A
 366 `int speed;` A

367 `int cfgetispeed (termios_p)` C
 368 `struct termios *termios_p;` A

369 `int cfsetispeed (termios_p, speed)` C
 370 `struct termios *termios_p;` A
 371 `int speed;` A

372 The *termios_p* argument is a pointer to a *termios* structure. C

373 *cfgetospeed()* returns the output baud rate stored in *c_cflag* pointed to by *termios_p*. C

374 *cfsetospeed()* sets the output baud rate stored in the *c_cflag* pointed to by *termios_p* to C
 375 *speed*. The zero baud rate, B0, is used to terminate the connection. If B0 is specified, 8
 376 the modem control lines shall no longer be asserted. Normally, this will disconnect the 8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

377 line. 8

378 *cfgetispeed()* returns the input baud rate stored in *c_cflag*. C

379 *cfsetispeed()* sets the input baud rate stored in *c_cflag* to *speed*. If the input baud rate is 8
380 set to zero, the input baud rate will be specified by the value of the output baud rate. For 8
381 any particular hardware, unsupported baud rate changes are ignored. This refers both to 8
382 changes to baud rates not supported by the hardware, and to changes setting the input and 8
383 output baud rates to different values if the hardware does not support this. 8

384 The CSIZE bits specify the character size in bits for both transmission and reception. 8
385 This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, 8
386 otherwise one stop bit. For example, at 110 baud, two stop bits are normally used. 8

387 If CREAD is set, the receiver is enabled. Otherwise, no characters shall be received. 8

388 If PARENB is set, parity generation and detection is enabled and a parity bit is added to 8
389 each character. If parity is enabled, PARODD specifies odd parity if set, otherwise even 8
390 parity is used. 8

391 If HUPCL is set, the modem control lines for the port shall be lowered when the last 8
392 process with the port open closes the port or the process terminates. The modem 8
393 connection shall be broken. If HUPCL is not set, the control lines are not altered. 8

394 If CLOCAL is set, a connection does not depend on the state of the modem status lines. If 8
395 CLOCAL is clear, the modem status lines shall be monitored. 8

396 Under normal circumstances, a call to the *open()* function shall wait for the modem 8
397 connection to complete. However, if the O_NONBLOCK flag is set (see *open()* §5.3.1) or 8
398 if CLOCAL has been set, the *open()* function shall return immediately without waiting 8
399 for the connection. For those files on which the connection has not been established, or 8
400 on which a modem disconnect has occurred, and for which CLOCAL is not set, both 8
401 *read()* and *write()* shall return a zero character count. For *read()*, this is equivalent to an 8
402 end-of-file condition. 8

403 If the object for which the control modes are set is not an asynchronous serial connection, 8
404 some of the modes may be ignored; e.g., if an attempt is made to set the baud rate on a 8
405 network connection to a terminal on another host, the baud rate may or may not be set on 8
406 the connection between that terminal and the machine it is directly connected to. 8

407 The initial hardware control value after *open()* is implementation defined. 8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

408 7.1.2.6 Local Modes 8

409 The *c_lflag* field of the argument structure is used to control various functions: 8

Mask Name	Description	
ECHO	Enable echo.	8
ECHOE	Echo ERASE as an error-correcting backspace.	B
ECHOK	Echo KILL.	B
ECHONL	Echo '\n'.	9
ICANON	Canonical input (erase and kill processing).	8
ISIG	Enable signals.	8
NOFLSH	Disable flush after interrupt, quit, or suspend.	8
TOSTOP	Send SIGTTOU for background output.	9

423 If ECHO is set, input characters are echoed back to the terminal. If ECHO is not set, input 8
 424 characters are not echoed. 8

425 If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase the c
 426 character from the display, if possible. c

427 If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase c
 428 the line from the display or shall echo the '\n' character after the KILL character. c

429 If ECHONL and ICANON are set, the '\n' character shall be echoed even if ECHO is not c
 430 set. c

431 If ISIG is set, each input character is checked against the special control characters INTR, c
 432 QUIT, and SUSP (Job Control Option only). If an input character matches one of these c
 433 control characters, the function associated with that character is performed. If ISIG is not c
 434 set, no checking is done. Thus these special input functions are possible only if ISIG is c
 435 set. c

436 If ICANON is set, canonical processing is enabled. This enables the erase and kill edit c
 437 functions, and the assembly of input characters into lines delimited by NL, EOF, and 9
 438 EOL, as described in Canonical Mode Input Processing §7.1.1.7. 8

439 If ICANON is not set, read requests are satisfied directly from the input queue. A read 8
 440 shall not be satisfied until at least MIN characters have been received or the timeout value B
 441 TIME expired between characters. The time value represents tenths of seconds. See the 8
 442 Non-Canonical Mode Input Processing §7.1.1.8 section for more details. 8
 443 9

444 If NOFLSH is set, the normal flush of the input and output queues associated with the 8
 445 INTR, QUIT, and SUSP (Job Control Option only) characters shall not be done. 8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

446 If TOSTOP (Job Control Option only) is set, the signal SIGTTOU is sent to the process A
 447 group of a process that tries to write to its controlling terminal if it is not in the A
 448 distinguished process group for that terminal. This signal, by default, stops the members A
 449 of the process group. Otherwise, the output generated by that process is output to the A
 450 current output stream. Processes that are blocking or ignoring SIGTTOU signals are C
 451 excepted and allowed to produce output and the SIGTTOU signal is not sent. C

452 The initial local control value after *open()* is implementation defined. 9

453 7.1.2.7 Special Control Characters 8

454 The special control characters values are defined by the array *c_cc*. The subscript name 8
 455 and description for each element in both canonical and non-canonical modes are as B
 456 follows: B

<u>Canonical</u> <u>Subscript</u>	<u>Non-Canonical</u> <u>Subscript</u>	<u>Description</u>	
VEOF		EOF character	B
VEOL		EOL character	B
VERASE		ERASE character	B
VINTR	VINTR	INTR character	B
VKILL		KILL character	B
	VMIN	MIN value	B
VQUIT	VQUIT	QUIT character	B
VSUSP	VSUSP	SUSP character	B
	VTIME	TIME value	B

469 The subscript values shall be unique, except that the VMIN and VTIME subscripts may B
 470 have the same values as the VEOF and VEOL subscripts, respectively. B

471 The VSUSP index shall be defined only if the Job Control Option is supported. 8

472 The number of elements in the *c_cc* array, NCCS, is implementation defined. C

473 The initial values of all control characters are implementation defined. 8

474 If `{_POSIX_V_DISABLE}` is in effect for the terminal file, and the value of one of the B
 475 special control characters is `{_POSIX_V_DISABLE}`, that function shall be disabled. The B
 476 `{_POSIX_V_DISABLE}` character is always read if received, and never causes a special B
 477 character function. B

478	7.2 General Terminal Interface Control Functions	8
479	The functions that are used to control the general terminal function are described in this	c
480	section. If the implementation supports the Job Control Option, unless otherwise noted	c
481	for a specific command, these functions are restricted from use by background processes.	B
482	Attempts to perform these operations shall cause the process group to be sent a SIGTTOU	B
483	signal. If the calling process is blocking or ignoring SIGTTOU signals, the process is	c
484	allowed to perform the operation and the SIGTTOU signal is not sent.	c
485	In all the functions, <i>fildev</i> is an open file descriptor.	8
486	7.2.1 Get and Set State	8
487	Functions: <i>tcgetattr()</i> , <i>tcsetattr()</i>	8
488	7.2.1.1 Synopsis	8
489	#include <termios.h>	A
490	int tcgetattr (<i>fildev</i> , <i>termios_p</i>)	A
491	int <i>fildev</i> ;	A
492	struct termios * <i>termios_p</i> ;	A
493	int tcsetattr (<i>fildev</i> , <i>optional_actions</i> , <i>termios_p</i>)	A
494	int <i>fildev</i> ;	A
495	int <i>optional_actions</i> ;	A
496	struct termios * <i>termios_p</i> ;	A
497	7.2.1.2 Description	8
498	The <i>tcgetattr()</i> function shall get the parameters associated with the object referred to by	8
499	<i>fildev</i> and store them in the <i>termios</i> structure referenced by <i>termios_p</i> . This command is	8
500	allowed from a background process; however, the information may be subsequently	8
501	changed by a foreground process.	8
502	The <i>tcsetattr()</i> function shall set the parameters associated with the terminal from the	8
503	<i>termios</i> structure referenced by <i>termios_p</i> as follows:	8
504	If <i>optional_actions</i> is TCSANOW, the change shall occur immediately.	A
505	If <i>optional_actions</i> is TCSADRAIN, the change shall occur after all output written	A
506	to <i>fildev</i> has been transmitted. This function should be used when changing	8
507	parameters that affect output.	8
508	If <i>optional_actions</i> is TCSADFLUSH, the change shall occur after all output	A
509	written to the object referred to by <i>fildev</i> has been transmitted, and all input that	8
510	has been received but not read shall be discarded before the change is made.	A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

511	7.2.1.3 Returns		A
512	Upon successful completion, a value of zero is returned. Otherwise, a value of -1 is		A
513	returned and <i>errno</i> is set to indicate the error.		A
514	7.2.1.4 Errors		A
515	If any of the following conditions occur, the <i>tcgetattr()</i> function shall return -1 and set		B
516	<i>errno</i> to the corresponding value:		B
517	[EBADF] The <i>fildev</i> argument is not a valid file descriptor.		A
518	[EINVAL] The device does not support the <i>tcgetattr()</i> function.		A
519	[ENOTTY] The file associated with <i>fildev</i> is not a terminal		C
520			B
521	If any of the following conditions occur, the <i>tcsetattr()</i> function shall return -1 and set		B
522	<i>errno</i> to the corresponding value:		B
523	[EBADF] The <i>fildev</i> argument is not a valid file descriptor.		A
524	[EINVAL] The device does not support the <i>tcsetattr()</i> function, or the		A
525	<i>optional_actions</i> argument is not a proper value.		A
526	[ENOTTY] The file associated with <i>fildev</i> is not a terminal		C
527			B
528	7.2.1.5 References		A
529	<termios.h> §7.1.2.		A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

530	7.2.2 Line Control Functions	8
531	Functions: <i>tcsendbreak()</i> , <i>tcdrain()</i> , <i>tcflush()</i> , <i>tcflow()</i>	8
532	7.2.2.1 Synopsis	8
533	#include <termios.h>	A
534	int tcsendbreak (<i>fildes</i> , <i>duration</i>)	A
535	int <i>fildes</i> ;	A
536	int <i>duration</i> ;	A
537	int tcdrain (<i>fildes</i>)	A
538	int <i>fildes</i> ;	A
539	int tcflush (<i>fildes</i> , <i>queue_selector</i>)	A
540	int <i>fildes</i> ;	A
541	int <i>queue_selector</i> ;	A
542	int tcflow (<i>fildes</i> , <i>action</i>)	A
543	int <i>fildes</i> ;	A
544	int <i>action</i> ;	A
545	7.2.2.2 Description	8
546	The <i>tcsendbreak()</i> function shall send a “break”; that is, a continuous stream of zero-	C
547	valued bits for a specific duration. If <i>duration</i> is zero, it shall send zero-valued bits for	C
548	0.25 seconds. If <i>duration</i> is not zero, it shall send zero-valued bits for an implementation	C
549	defined period of time.	C
550	The <i>tcdrain()</i> function shall wait until all output written to the object referred to by <i>fildes</i>	8
551	has been transmitted.	8
552	The <i>tcflush()</i> function shall discard data written to the object referred to by <i>fildes</i> but not	8
553	transmitted, or data received but not read, depending on the value of <i>queue_selector</i> :	8
554	If <i>queue_selector</i> is TCIFLUSH, it shall flush data received but not read.	8
555	If <i>queue_selector</i> is TCOFLUSH, it shall flush data written but not transmitted.	8
556	If <i>queue_selector</i> is TCIOFLUSH, it shall flush both data received but not read,	8
557	and data written but not transmitted.	8
558	The <i>tcflow()</i> function shall suspend transmission or reception of data on the object	8
559	referred to by <i>fildes</i> , depending on the value of <i>action</i> :	8
560	If <i>action</i> is TCOOFF, it shall suspend output.	8
561	If <i>action</i> is TCOON, it shall restart suspended output.	8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

562		If <i>action</i> is TCIOFF, it shall suspend input.	B
563		If <i>action</i> is TCION, it shall restart suspended input.	B
564		The default on open of a terminal file is that neither its input nor its output are suspended.	B
565	7.2.2.3	Returns	A
566		Upon successful completion, a value of zero is returned. Otherwise, a value of -1 is	A
567		returned and <i>errno</i> is set to indicate the error.	A
568	7.2.2.4	Errors	A
569		If any of the following conditions occur, the <i>tcsendbreak()</i> function shall return -1 and	B
570		set <i>errno</i> to the corresponding value:	B
571	[EBADF]	The <i>fildev</i> argument is not a valid file descriptor.	A
572	[EINVAL]	The device does not support the <i>tcsendbreak()</i> function.	A
573	[ENOTTY]	The file associated with <i>fildev</i> is not a terminal	C
574		If any of the following conditions occur, the <i>tcdrain()</i> function shall return -1 and set	B
575		<i>errno</i> to the corresponding value:	B
576	[EBADF]	The <i>fildev</i> argument is not a valid file descriptor.	A
577	[EINTR]	A signal interrupted the <i>tcdrain()</i> function.	C
578	[EINVAL]	The device does not support the <i>tcdrain()</i> function.	A
579	[ENOTTY]	The file associated with <i>fildev</i> is not a terminal	C
580		If any of the following conditions occur, the <i>tcflush()</i> function shall return -1 and set	B
581		<i>errno</i> to the corresponding value:	B
582	[EBADF]	The <i>fildev</i> argument is not a valid file descriptor.	A
583	[EINVAL]	The device does not support the <i>tcflush()</i> function, or the	A
584		<i>queue_selector</i> argument is not a proper value.	A
585	[ENOTTY]	The file associated with <i>fildev</i> is not a terminal	C
586		If any of the following conditions occur, the <i>tcflow()</i> function shall return -1 and set	B
587		<i>errno</i> to the corresponding value:	B
588	[EBADF]	The <i>fildev</i> argument is not a valid file descriptor.	A
589	[EINVAL]	The device does not support the <i>tcflow()</i> function, or the <i>action</i>	A
590		argument is not a proper value.	A
591	[ENOTTY]	The file associated with <i>fildev</i> is not a terminal	C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

592	7.2.2.5 References	B
593	<termios.h> §7.1.2.	B
594		B
595	7.2.3 Get Distinguished Process Group ID	B
596	Function: <i>tcgetpgrp()</i>	B
597	7.2.3.1 Synopsis	B
598	#include <termios.h>	A
599	int tcgetpgrp (<i>fildes</i>)	A
600	int <i>fildes</i> ;	A
601	7.2.3.2 Description	B
602	The <i>tcgetpgrp()</i> function shall be provided if the implementation supports the Job	B
603	Control Option.	B
604	The <i>tcgetpgrp()</i> function shall return the value of the process group ID of the	B
605	distinguished process group associated with the terminal.	B
606	The <i>tcgetpgrp()</i> function is allowed from a background process; however, the	B
607	information may be subsequently changed by a foreground process.	B
608	7.2.3.3 Returns	B
609	Upon successful completion, <i>tcgetpgrp()</i> returns the process group ID of the	B
610	distinguished process group associated with the terminal. Otherwise, a value of -1 is	B
611	returned and <i>errno</i> is set to indicate the error.	B
612	7.2.3.4 Errors	B
613	If any of the following conditions occur, the <i>tcgetpgrp()</i> function shall return -1 and set	B
614	<i>errno</i> to the corresponding value:	B
615	[EBADF] The <i>fildes</i> argument is not a valid file descriptor.	B
616	[EINVAL] This function is not allowed for the device associated with the	B
617	<i>fildes</i> argument.	B
618	[ENOTTY] The calling process does not have a controlling terminal or the file	C
619	is not the controlling terminal.	C
620	7.2.3.5 References	B
621	<i>setpgrp()</i> §4.3.2, <i>jcsetpgrp()</i> §4.3.3, <i>tcsetpgrp()</i> §7.2.4.	B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

622	7.2.4 Set Distinguished Process Group ID		B
623	Function: <i>tcsetpgrp()</i>		B
624	7.2.4.1 Synopsis		B
625	<code>#include <termios.h></code>		B
626	<code>int tcsetpgrp (<i>fildev</i>, <i>pgrp_id</i>)</code>		B
627	<code>int <i>fildev</i>;</code>		B
628	<code>int <i>pgrp_id</i>;</code>		B
629	7.2.4.2 Description		B
630	The <i>tcsetpgrp()</i> function shall be provided if the implementation supports the Job		B
631	Control Option.		B
632	If the process has a controlling terminal, the <i>tcsetpgrp()</i> function shall set the		B
633	distinguished process group ID associated with the terminal to <i>pgrp_id</i> . The file		B
634	associated with <i>fildev</i> must be the controlling terminal of the calling process. There must		B
635	be at least one process in <i>pgrp_id</i> that has the same controlling terminal as the calling		B
636	process.		B
637	7.2.4.3 Returns		B
638	Upon successful completion, <i>tcsetpgrp()</i> returns a value of zero. Otherwise, a value of		B
639	-1 is returned and <i>errno</i> is set to indicate the error.		B
640	7.2.4.4 Errors		B
641	If any of the following conditions occur, the <i>tcsetpgrp()</i> function shall return -1 and set		B
642	<i>errno</i> to the corresponding value:		B
643	[EBADF] The <i>fildev</i> argument is not a valid file descriptor.		B
644	[EINVAL] This function is not allowed for the device associated with the		C
645	<i>fildev</i> argument or the value of the <i>pgrp_id</i> argument is less than or		C
646	equal to zero, or exceeds {PID_MAX}.		C
647	[ENOTTY] The calling process does not have a controlling terminal or the file		B
648	is not the controlling terminal.		B
649	[EPERM] The value of the <i>pgrp_id</i> argument is greater than zero and less		B
650	than or equal to {PID_MAX}, and there is no process in the process		B
651	group indicated by <i>pgrp_id</i> that has the same controlling terminal		B
652	as the calling process.		B
653			B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

8. C Language Library

1 8.1 Referenced C Language Routines

- 2 When the *ANSI/X3.159-198x Programming Language C Standard* is adopted, it will be **C**
3 the basis for a C language binding to IEEE Std 1003.1. In the interim, the following **B**
4 routines are left unstandardized, but are defined by common usage and traditional **B**
5 implementations. Although the lack of an adopted C language standard negatively **B**
6 affects the ability of applications developers to write portable applications, they can use **B**
7 draft versions of the *ANSI/X3.159-198x Programming Language C Standard* and **B**
8 common usage as guidance to maximize the future portability of their applications. **B**
- 9 • 4.2 Diagnostics
10 Functions: assert.
 - 11 • 4.3 Character Handling
12 Functions: isalnum, isalpha, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace,
13 isupper, isxdigit, tolower, toupper.
 - 14 • 4.5 Mathematics
15 Functions: acos, asin, atan, atan2, cos, sin, tan, cosh, sinh, tanh, exp, frexp, ldexp, **C**
16 log, log10, modf, pow, sqrt, ceil, fabs, floor, fmod.
 - 17 • 4.6 Non-Local Jumps
18 Functions: setjmp, longjmp.
 - 19 • 4.7 Signal Handling
20 Functions: signal+. **B**
 - 21 • 4.9 Input/Output
22 Functions: clearerr, fclose, feof, ferror, fflush, fgetc, fgets, fopen, fputc, fputs, fread,
23 freopen, fseek, ftell, fwrite, getc, getchar, gets, perror, printf, fprintf, sprintf, putc, **B**
24 putchar, puts, remove, rename+, rewind, scanf, fscanf, sscanf, setbuf, tmpfile,
25 tmpnam, ungetc.
 - 26 • 4.10 General Utilities
27 Functions: abs, atof, atoi, atol, rand, srand, calloc, free, malloc, realloc, abort, exit, **C**
28 getenv+, bsearch, qsort, setlocale+. **C**

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 29 • 4.11 String Handling
 30 Functions: *strcpy*, *strncpy*, *strcat*, *strncat*, *strcmp*, *strchr*, *strcspn*, *strpbrk*, *strchr*,
 31 *strspn*, *strstr*, *strtok*, *strlen*.
- 32 • 4.12 Date and Time
 33 Functions: *time+*, *asctime+*, *ctime+*, *gmtime+*, *localtime+*, *strftime+*. c
- 34 Functions indicated above with a + are included in both documents. Descriptions of
 35 these routines have been retained in this standard because they represent further c
 36 specifications or amplifications of the versions defined by the *ANSI/X3.159-198x* 8
 37 *Programming Language C Standard*. 8
- 38 Systems conforming to the IEEE Std 1003.1 shall make no distinction between the “text 8
 39 streams” and the “binary streams” described in the *ANSI/X3.159-198x Programming* 8
 40 *Language C Standard*. 8
- 41 For the *fseek()* function, if the specified position is beyond end-of-file, the consequences 8
 42 described in *lseek()* (see *lseek()* §6.5.3) shall occur. 8
- 43 8
- 44 **8.1.1 Extensions to *asctime()* Function**
- 45 If the environment variable named TZ is present, (see Environment Variables §2.7) the c
 46 functions *asctime()*, *strftime()*, *localtime()*, *ctime()*, and *gmtime()* use its contents to c
 47 override the default time zone. The value of TZ has the form (spaces inserted for c
 48 clarity): c
- 49 *std offset dst offset ,rule* c
- 50 or in an expanded format: c
- 51 *stdoffset[dst[offset][,start[/time],end[/time]]]* c
 52 */characters* c
- 53 Where: c
- 54 If the first character of the environment variable TZ is a slash (/), it is assumed the c
 55 characters following the slash are handled in an implementation defined manner. c
- 56 *std* and *dst* c
- 57 Three or more bytes that are the designation for the standard (*std*) c
 58 or summer (*dst*) time zone. Only *std* is required; if *dst* is missing, c
 59 then summer time does not apply in this locale. Upper- and c
 60 lowercase letters are explicitly allowed. Any characters except c
 61 digits, comma (,), minus (-), plus (+), and ASCII NUL are allowed. c
- 62 *offset* c
- 63 Indicates how far west (or, if preceded by “-”, east) of Greenwich c
 64 that time zone lies. The *offset* has the form: c

65		<i>hh</i> [: <i>mm</i> [: <i>ss</i>]]	c
66		The minutes (<i>mm</i>) and seconds (<i>ss</i>) are optional. The hour (<i>hh</i>)	c
67		shall be required and may be a single digit. The <i>offset</i> following	c
68		<i>std</i> shall be required. If no <i>offset</i> follows <i>dst</i> , summer time is	c
69		assumed to be one hour ahead of standard time. One or more	c
70		digits may be used; the value is always interpreted as a decimal	c
71		number. The hour shall be between 0 and 12, and the minutes (and	c
72		seconds) — if present — between 0 and 59. Out of range values	c
73		may cause unpredictable behavior. If preceded by a “-”, the time	c
74		zone shall be east of Greenwich, otherwise it shall be west (which	c
75		may be indicated by an optional preceding “+”).	c
76	<i>rule</i>		c
77		Indicates when to change to and back from summer time. The <i>rule</i>	c
78		has the form:	c
79		<i>date / time , date / time</i>	c
80		where the first <i>date</i> describes when the change from standard to	c
81		summer time occurs and the second <i>date</i> describes when the	c
82		change back happens. Each <i>time</i> field describes when, in current	c
83		local time, the change to the other time is made.	c
84		The format of <i>date</i> shall be one of the following:	c
85		<i>Jn</i>	c
86		The Julian day <i>n</i> ($1 \leq n \leq 365$). Leap days	c
87		shall not be counted. That is, in all years —	c
88		including leap years — February 28 is day	c
89		59 and March 1 is day 60. It is impossible	c
90		to explicitly refer to the occasional February	c
91		29.	c
92		<i>n</i>	c
93		The zero-based Julian day ($0 \leq n \leq 365$).	c
94		Leap days shall be counted, and it is	c
95		possible to refer to February 29.	c
96		<i>Mm.n.d</i>	c
97		The <i>d</i> th day ($0 \leq d \leq 6$) of week <i>n</i> of month	c
98		<i>m</i> of the year ($1 \leq n \leq 5$, $1 \leq m \leq 12$, where	c
99		week 5 means “the last <i>d</i> day in month <i>m</i> ”	c
100		which may occur in either the fourth or the	c
101		fifth week).	c
102		The <i>time</i> has the same format as <i>offset</i> except that no leading sign (“-” or	c
103		“+”) shall be allowed. The default, if <i>time</i> is not given, shall be	c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

104 `02:00:00.` c

105 If no *rule* is specified and summer time applies, United States federal law c
106 shall be assumed. c

107 If the first character of the *rule* is a slash (/), the bytes following the slash c
108 shall be handled in an implementation defined manner. c

109 The effects of setting TZ are, thus, to change the values of the external variable `timezone` c
110 and `daylight`. In addition, the time zone names contained in the external variable c

111 `char *tzname[2] = {"std", "dst"};` c

112 are set from the environment variable TZ. c

113 It is explicitly allowed for programs to change TZ and have the changed TZ apply to c
114 themselves. c

115 **8.1.2 Extensions to *setlocale()* Function** c

116 Function: *setlocale()* c

117 **8.1.2.1 Synopsis** c

118 `char *setlocale (category, locale)` c
119 `int category;` c
120 `char *locale;` c

121 **8.1.2.2 Description** c

122 The *ANSI/X3.159-198x Programming Language C Standard* allows the specification of c
123 an implementation defined native environment for the *setlocale()* function, which will c
124 set a specific category to an implementation defined default. For IEEE Std 1003.1 c
125 systems, this corresponds to the value of the environment variables. c

126 Setting a specific category to an implementation defined default is invoked by setting the c
127 *locale* argument to point to a null string, and by setting the *category* argument to one of c
128 the integer values: c

129 `LC_CTYPE` c
130 `LC_COLLATE` c
131 `LC_TIME` c
132 `LC_NUMERIC` c

133 In all cases, *setlocale()* will first check the value of the corresponding environment c
134 variable (e.g., `LC_CTYPE` for the `LC_CTYPE` category) and if valid (i.e., points to the c
135 name of a valid locale), *setlocale()* will set the specified category of the international c
136 environment to that value and return the string corresponding to the locale set (i.e., the c
137 value of the environment variable, not ""). If the value is invalid, *setlocale()* will c
138 return a null pointer and the international environment is not changed. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

139 If the environment variable corresponding to the specified category is not set or is set to c
 140 the empty string, the behavior of *setlocale()* is implementation defined, unless the LANG c
 141 environment variable is set and valid in which case *setlocale()* will set the category to c
 142 the corresponding value of LANG. In some implementations, this may default to a c
 143 system-wide value, others may default to the "C" locale. Setting all categories to the c
 144 implementation defined default is similar to the previous usage, but it interrogates all the c
 145 environment variables to determine the specific value to set. To set all categories in the c
 146 international environment, *setlocale()* is invoked in the following manner: c

```
147         setlocale(LC_ALL, ""); c
```

148 To satisfy this request, *setlocale()* first checks all the environment variables. If any c
 149 environment variable is invalid, *setlocale()* returns a null pointer and the international c
 150 environment is not changed. c

151 If they are valid, *setlocale()* sets the international environment to reflect the values of the c
 152 environment variables. The categories are set in the following order: c

```
153         LC_ALL c  

  154         LC_CTYPE c  

  155         LC_COLLATE c  

  156         LC_TIME c  

  157         LC_NUMERIC c  

  158         new categories c
```

159 Using this scheme, the categories corresponding to the environment variables will c
 160 override the value of the LANG environment variable for a particular category. c

161 If one or all of the category-specific environment variables (i.e., LC_CTYPE, c
 162 LC_COLLATE, LC_TIME, or LC_NUMERIC) are not set, the particular category is not c
 163 overridden. If one or all of the category-specific environment variables are set to the c
 164 empty string, the behavior is implementation defined. c

165 If the LANG environment variable is not set or is set to the empty string, the behavior of c
 166 *setlocale()* is implementation defined. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

167 **8.2 FILE-Type C Language Functions**

168 This section describes functions which make reference to the FILE type, as described in **B**
 169 the *ANSI/X3.159-198x Programming Language C Standard*. **B**

170 **8.2.1 Map a Stream Pointer to a File Descriptor**171 Function: *fileno()*172 **8.2.1.1 Synopsis**173 `#include <stdio.h>`174 `int fileno (stream)`175 `FILE *stream;`176 **8.2.1.2 Description**

177 The *fileno()* function returns the integer file descriptor associated with the *stream* (see
 178 *open()* §5.3.1).

179 There is a fixed relationship between the C language *stdin*, *stdout*, and *stderr* and the **c**
 180 initial corresponding file descriptor values. The following symbolic values in **c**
 181 `<unistd.h>` §2.10 define this relationship: **c**

182 `STDIN_FILENO` Standard input value, *stdin*. **c**

183 `STDOUT_FILENO` Standard output value, *stdout*. **c**

184 `STDERR_FILENO` Standard error value, *stderr*. **c**

185 **8.2.1.3 References**186 *open()* §5.3.1.

187 8.2.2 Open a Stream on a File Descriptor

188 Function: *fdopen()*

189 8.2.2.1 Synopsis

```
190         #include <stdio.h>
191         FILE *fdopen (fildes, type)
192         int fildes;
193         char *type;
```

194 8.2.2.2 Description

195 The *fdopen()* routine associates a stream with a file descriptor.196 The *type* argument is a character string having one of the following values:

197	" r "	open for reading
198	" w "	open for writing
199	" a "	open for writing at end of file
200	" r+ "	open for update (reading and writing)
201	" w+ "	open for update (reading and writing)
202	" a+ "	open for update (reading and writing) at end of file

203 The types **r+**, **w+**, and **a+** are equivalent, except that **a+** implicitly seeks to the end of
204 the file.205 Additional values for the *type* argument may be defined by an implementation.206 The *type* of the stream must be allowed by the mode of the open file.

207 8.2.2.3 Returns

208 If successful, the *fdopen()* function returns a pointer to a stream. Otherwise, a NULL
209 pointer is returned.

210 8.2.2.4 References

211 *open()* §5.3.1, *fopen()* (ANSI/X3.159-198x Programming Language C Standard).

8

212	8.3 Other C Language Functions	B
213	8.3.1 Non-Local Jumps	8
214	Functions: <i>sigsetjmp()</i> , <i>siglongjmp()</i>	B
215	8.3.1.1 Synopsis	B
216	#include <setjmp.h>	B
217	int sigsetjmp (env, savemask)	B
218	sigjmp_buf env;	B
219	int savemask;	B
220	void siglongjmp (env, val)	B
221	sigjmp_buf env;	B
222	int val;	B
223	8.3.1.2 Description	B
224	The <i>sigsetjmp()</i> macro shall comply with the definition of the <i>setjmp()</i> macro in the	C
225	<i>ANSI/X3.159-198x Programming Language C Standard</i> . If the value of the <i>savemask</i>	B
226	argument is not zero, the <i>sigsetjmp()</i> function shall also save the process's current signal	B
227	mask (see <signal.h> §3.3.1) as part of the calling environment.	B
228	The <i>siglongjmp()</i> function shall comply with the definition of the <i>longjmp()</i> function in	B
229	the <i>ANSI/X3.159-198x Programming Language C Standard</i> . If and only if the <i>env</i>	B
230	argument was initialized by a call to the <i>sigsetjmp()</i> function with a non-zero <i>savemask</i>	B
231	argument, the <i>siglongjmp()</i> function shall restore the saved signal mask.	B
232	8.3.1.3 References	8
233	<i>sigaction()</i> §3.3.4, <signal.h> §3.3.1, <i>sigprocmask()</i> §3.3.5, <i>sigsuspend()</i> §3.3.7.	8

234	8.3.2 Specify Signal Handling	c
235	Function: <i>signal()</i>	c
236	8.3.2.1 Synopsis	c
237	#include <signal.h>	c
238	void (*signal (<i>sig</i> , <i>func</i>)) ()	c
239	int <i>sig</i> ;	c
240	void (* <i>func</i>) ();	c
241	8.3.2.2 Description	c
242	The <i>ANSI/X3.159-198x Programming Language C Standard</i> defines the <i>signal()</i> function	c
243	as a means of specifying the action to be taken upon receipt of a signal.	c
244	In general, the use of the <i>signal()</i> function shall not conflict with the behavior of signals	c
245	as characterized in this standard. However, there may be implementation defined side	c
246	effects associated with the use of the <i>signal()</i> function. For instance, if the <i>signal()</i>	c
247	function is invoked to establish a signal-catching function or to set the action to SIG_DFL	c
248	while the signal is pending, the pending signal may be discarded (unless the signal is	c
249	SIGKILL or SIGSTOP).	c
250	The <i>sigaction()</i> §3.3.4 function provides an alternative interface that assures the delivery	c
251	of signals and the integrity of signal-catching functions.	c
252	The <i>sigaction()</i> function shall properly return, in the structure pointed to by <i>oact</i> , the	c
253	previous signal action, even if that action had been established by the <i>signal()</i> function.	c
254	In such a case, the values of the fields of the structure pointed to by <i>oact</i> are undefined,	c
255	and in particular <i>oact->sv_handler</i> is not necessarily the same value passed to the	c
256	<i>signal()</i> function. However, if a pointer to the structure is passed to a subsequent call to	c
257	the <i>sigaction()</i> function via the <i>act</i> parameter, handling of the signal shall be reinstated	c
258	as if the original call to the <i>signal()</i> function were repeated.	c
259	It is implementation defined whether the return value of the <i>signal()</i> function will	c
260	accurately reflect the previous signal action if that action had been established by the	c
261	<i>sigaction()</i> function. It is also implementation defined whether a signal mask established	c
262	by the <i>sigaction()</i> function is preserved when the signal action for that signal is altered	c
263	by the <i>signal()</i> function. Because of this unpredictability, the <i>sigaction()</i> and <i>signal()</i>	c
264	functions should not be used in the same process to control the same signal.	c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.



9. System Databases

C

1 9.1 System Databases

2 The routines described in this section allow an application to access the two system
3 databases that are described below.

4 The *group* database contains the following information for each group:

5 group name

6

B

7 numerical group ID

8 list of the names or numbers of all users allowed in the group

9 The *passwd* database contains the following information for each user:

10 login name

11

B

12 numerical user ID

13 numerical group ID

14 initial working directory

15 initial user program

16 If the initial program field is null, the system default is used.

17 If the initial working directory field is null, the interpretation of that field is
18 implementation defined.

19

B

20 These databases may contain other fields that are implementation defined.

21 9.2 Database Access

22 9.2.1 Group Database Access

23 Functions: *getgrent()*, *getgrgid()*, *getgrnam()*, *setgrent()*, *endgrent()*

24 9.2.1.1 Synopsis

25 `#include <grp.h>`26 `struct group *getgrent ()`27 `struct group *getgrgid (gid)`28 `uid_t gid;` 829 `struct group *getgrnam (name)`30 `char *name;`31 `void setgrent ()`32 `void endgrent ()`

33 9.2.1.2 Description

34 The *getgrent()*, *getgrgid()* and *getgrnam()* routines each return pointers to an object of
 35 type *struct group* containing an entry from the group database. The members of this
 36 structure, which is defined in *<grp.h>*, include:

<u>Member Type</u>	<u>Member Name</u>	<u>Description</u>	
<i>char *</i>	<i>gr_name</i>	The name of the group	B
<i>uid_t</i>	<i>gr_gid</i>	The numerical group ID	8
<i>char **</i>	<i>gr_mem</i>	A null-terminated vector of pointers to the individual member names	

44 The *getgrent()* function reads the next entry of the database, so successive calls shall
 45 search the entire database. The *getgrgid()* and *getgrnam()* functions search from the
 46 beginning of the database until a matching *gid* or *name* is found, or the end of the
 47 database is encountered.

48 A call to *setgrent()* has the effect of rewinding the group database to allow repeated
 49 searches. A call to the *endgrent()* function should be used to close the group database
 50 when processing is complete.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

51 9.2.1.3 Returns

52 A NULL pointer is returned on error or when the end of the database is encountered.

53 The return values may point to static data that is overwritten by each call. 9

54 9.2.1.4 References

55 *getlogin()* §4.2.4, *getpwent()* §9.2.2.

56 9.2.2 User Database Access C

57 Functions: *getpwent()*, *getpwuid()*, *getpwnam()*, *setpwent()*, *endpwent()*

58 9.2.2.1 Synopsis

59 `#include <pwd.h>`60 `struct passwd *getpwent ()`61 `struct passwd *getpwuid (uid)`62 `uid_t uid;` 863 `struct passwd *getpwnam (name)`64 `char *name;`65 `void setpwent ()`66 `void endpwent ()`

67 9.2.2.2 Description

68 The *getpwent()*, *getpwuid()* and *getpwnam()* functions each return a pointer to an object
69 of type *struct passwd* containing an entry from the user database. The members of this C
70 structure, which is defined in `<pwd.h>`, include:

<u>Member Type</u>	<u>Member Name</u>	<u>Description</u>	
<i>char *</i>	<i>pw_name</i>	User's login name	B
<i>uid_t</i>	<i>pw_uid</i>	User ID number	8
<i>uid_t</i>	<i>pw_gid</i>	Group ID number	8
<i>char *</i>	<i>pw_dir</i>	Home Directory	
<i>char *</i>	<i>pw_shell</i>	Default shell	

79 The *struct passwd* structure used by these routines may include additional members. The
80 additional member names shall be declared in `<pwd.h>` and shall begin with the prefix
81 "pw_". 982 The *getpwent()* function reads the next entry in the database, so successive calls can be
83 used to search the entire database. The *getpwuid()* and *getpwnam()* functions search
84 from the beginning of the database until a matching *uid* or *name* is found, or the end of

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 85 the database is encountered.
- 86 A call to *setpwent()* has the effect of rewinding the user database to allow repeated c
87 searches. A call to *endpwent()* closes the password database when processing is
88 complete.
- 89 The implementation of the *cuserid()* §4.2.4 function may use the *getpwnam()* function; 9
90 thus the results of a user's call to either routine may be overwritten by a subsequent call 9
91 to the other routine. 9
- 92 **9.2.2.3 Returns**
- 93 A NULL pointer is returned on error or the end of the database is encountered.
- 94 The return values may point to static data that is overwritten on each call. 9
- 95 **9.2.2.4 References**
- 96 *cuserid()* §4.2.4, *getlogin()* §4.2.4, *getgrent()* §9.2.1.
- 97 c

10. Data Interchange Format

1 10.1 Archive/Interchange File Format

B

2 A conforming system shall provide a mechanism to copy files from a medium to the local B
3 file system and copy files from the local file system to a medium using the interchange B
4 format described here. This standard does not define this mechanism.* C

5 When this mechanism is used to copy files from the medium by a nonprivileged process, B
6 the protection information (ownership and access permissions) shall be set in the same B
7 fashion that *creat()* §5.3.2 would when given the mode argument matching the file B
8 permissions supplied by the *mode* field of this format. B

9 The *format-creating utility* is used to translate from the file system to the formats defined C
10 in this section, in an implementation defined way, and the *format-reading utility* is used C
11 to translate from the formats defined in this section to a file system. C

12 10.1.1 *cpio* Archive Format

B

13 The byte-oriented *cpio* archive format is a series of entries, each comprised of a header B
14 that describes the file, the name of the file, and then the contents of the file. B

15 An archive may be recorded as a series of fixed size blocks of bytes. This blocking shall B
16 be used only to make physical I/O more efficient. The last group of blocks is always at B
17 the full size. B

18 For the byte-oriented *cpio* archive format, the individual entry information must be in B
19 the order indicated and is described by: B

* The P1003.2 Working Group is working on this mechanism. See Shell and Utilities §A.2.2.

Byte-Oriented *cpio* Archive Entry

Header			B
<u>Field Name</u>	<u>Length</u>	<u>Interpreted as</u>	B
<i>c_magic</i>	6 bytes	octal number	B
<i>c_dev</i>	6 bytes	octal number	B
<i>c_ino</i>	6 bytes	octal number	B
<i>c_mode</i>	6 bytes	octal number	B
<i>c_uid</i>	6 bytes	octal number	B
<i>c_gid</i>	6 bytes	octal number	B
<i>c_nlink</i>	6 bytes	octal number	B
<i>c_rdev</i>	6 bytes	octal number	B
<i>c_mtime</i>	11 bytes	octal number	B
<i>c_namesize</i>	6 bytes	octal number	B
<i>c_filesize</i>	11 bytes	octal number	B

File Name			B
<u>Field Name</u>	<u>Length</u>	<u>Interpreted as</u>	B
<i>c_name</i>	<i>c_namesize</i>	pathname string	B

File Data			B
<u>Field Name</u>	<u>Length</u>	<u>Interpreted as</u>	B
<i>c_filedata</i>	<i>c_filesize</i>	data	B

42 10.1.1.1 Header

43 For each file in the archive, a header as defined above shall be written. The information
 44 in the header fields shall be written as streams of bytes interpreted as octal numbers and
 45 shall be right-justified and zero filled. The fields shall be interpreted as follows:

- 46 • *c_magic* shall identify the archive as being a transportable archive by B
 47 containing the magic bytes as defined by MAGIC ("070707"). B
- 48 • *c_dev* and *c_ino* shall contain values which uniquely identify the file within B
 49 the archive (i.e., no files shall contain the same pair of *c_dev* and *c_ino* values B
 50 unless they are links to the same file). The values shall be determined in an B
 51 implementation defined manner. B
- 52 • *c_mode* shall contain the file type and access permissions as defined in the B
 53 tables below. B
- 54 • *c_uid* shall contain the user id of the owner. B
- 55 • *c_gid* shall contain the group id of the group. B
- 56 • *c_nlink* shall contain the number of links referencing the file at the time the B
 57 archive was created. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

- 58 • *c_rdev* shall contain implementation defined information for character or B
59 block special files. B
- 60 • *c_mtime* shall contain the latest time of modification of the file. B
- 61 • *c_namesize* shall contain the length of the path name, including the B
62 terminating null byte. B
- 63 • *c_filesize* shall contain the length of the file. This is the length of the data B
64 section following the header structure. B
- 65 **10.1.1.2 File Name** B
- 66 *c_name* shall contain the path name of the file. The length of the name is determined by B
67 *c_namesize*; the maximum length of this string is 256 bytes. B
- 68 **10.1.1.3 File Data** B
- 69 Following *c_name*, there shall be *c_filesize* bytes of data. Interpretation of such data B
70 shall occur in a manner dependent on the file. If *c_filesize* is zero, no data shall be B
71 contained in *c_filedata*. B
- 72 **10.1.1.4 Special Entries** B
- 73 Special files, directories, and the trailer are recorded with *c_filesize* equal to zero. The B
74 header for the next file entry in the archive shall be written directly after the last byte of B
75 the file entry preceding it. A header denoting the file name "TRAILER!!!" shall B
76 indicate the end of the archive; the contents of bytes in the last block of the archive B
77 following such a header are undefined. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

- 78 10.1.1.5 `cpio` Values B
 79 Values needed by the `cpio` archive format are described as follows: B

Values for `c_mode` field B

File permissions B

<u>Name</u>	<u>Value</u>	<u>Indicates</u>	B
<code>C_IRUSR</code>	000400	read by owner	B
<code>C_IWUSR</code>	000200	write by owner	B
<code>C_IXUSR</code>	000100	execute by owner	B
<code>C_IRGRP</code>	000040	read by group	B
<code>C_IWGRP</code>	000020	write by group	B
<code>C_IXGRP</code>	000010	execute by group	B
<code>C_IROTH</code>	000004	read by others	B
<code>C_IWOTH</code>	000002	write by others	B
<code>C_IXOTH</code>	000001	execute by others	B
<code>C_ISUID</code>	004000	set uid	B
<code>C_ISGID</code>	002000	set gid	B
<code>C_ISVTX</code>	001000	reserved	B

Values for `c_mode` field B

File type B

<u>Name</u>	<u>Value</u>	<u>Indicates</u>	B
<code>C_ISDIR</code>	040000	directory	B
<code>C_ISFIFO</code>	010000	FIFO	B
<code>C_ISREG</code>	100000	regular file	B
<code>C_ISBLK</code>	060000	block special	B
<code>C_ISCHR</code>	020000	character special	B
	110000	reserved	B
	120000	reserved	B
	140000	reserved	B

- 109 `C_ISDIR`, `C_ISFIFO`, and `C_ISREG` shall be supported on a IEEE Std 1003.1 conforming B
 110 system; additional values defined above are reserved for compatibility with existing B
 111 systems. Additional file types may be supported; however, such files should not be B
 112 written on archives intended for transport to portable systems. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

- 113 **10.1.1.6 References** B
114 `<grp.h>` §9.2.1, `<pwd.h>` §9.2.2, `<sys/stat.h>` §5.6.1, `chmod()` §5.6.4, `link()` §5.3.4, B
115 `mkdir()` §5.4.1, `read()` §6.4.1, `stat()` §5.6.2.
- 116 **10.1.2 Multiple Volumes**
117 It shall be possible for data represented by the Archive/Interface File Format to reside in c
118 more than one file. c
- 119 The format is considered a stream of bytes. Any two bytes may be separated by the end c
120 of a file. c
- 121 The end-of-file is used as an indicator that a new file is to be read, and the format-reading c
122 utility will, in an implementation defined manner, determine the next file. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

Appendices

1 (These appendices are not a part of IEEE Std 1003.1, IEEE Standard Portable Operating c
2 System Interface for Computer Environments.) c

3 A. Related Standards

4 This appendix describes other standards efforts, related to IEEE Std 1003.1, that are
5 available or under development.

6 A.1 Related Standards — Open System Architecture

7 This IEEE Std 1003.1 is intended to complement others that together would provide a
8 comprehensive Open System Architecture. The standards in these areas fall into three
9 areas: ones directly related to the IEEE Std 1003.1, ones already available and of use to
10 those interested in Open Systems Architectures, and finally, those in development.

11 IEEE and ANSI/IEEE standards can be ordered from:

12	IEEE Service Center	IEEE Computer Society
13	445 Hoes Lane	Box 80452, Worldway Postal Center
14	Piscataway, NJ 08854	Los Angeles, CA 90080
15	(201) 981-0060	(800) 272-6657
16		(714) 821-8380 in California

17 The document X3/SD-4 provides a list of all active X3 and related ISO projects, c
18 including approved standards. X3/SD-4 is available from: c

19	CBEMA
20	X3 Secretariat
21	311 First Street, NW Suite 500
22	Washington, DC 20001-2178
23	(202) 737-8888

24 ANSI and ISO standards can be ordered from: c

25	ANSI	c
26	1430 Broadway	c
27	New York, NY 10018	c
28	(212) 642-4900	c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

29 A.2 Standards Closely Related to the 1003.1 Document

30 A.2.1 C Language Standard

31 This document refers to the C Language Standard effort presently under development by
 32 Technical Committee X3J11 of the Accredited Standards Committee X3 — Information
 33 Processing Systems. The X3J11 and 1003.1 groups have been cooperating to insure that
 34 the standards are complementary and not overlapping. At the time of publication, the
 35 most recent X3J11 material was the version for public comment of the *ANSI/X3.159-198x*
 36 *Programming Language C Standard*, available from:

37 Global Engineering Documents, Inc.
 38 2625 Hickory Street
 39 Santa Ana, CA 92707
 40 (800) 854-7179
 41 (714) 540-9870

42 Once the X3J11 document is approved, it will be available from the ANSI address given
 43 above.

44 A.2.2 Shell and Utilities

45 This area is currently in development by IEEE Computer Society Working Group
 46 P1003.2. The proposed 1003.2 standard defines a source code level interface to shell
 47 services and common utility programs for application programs conforming to IEEE Std
 48 1003.1.* The proposed standard is being designed to be used by both application
 49 programmers and system implementors.

50 The following goals have been established for the Working Group:

51 Specify a standard interface that may be accessed in common by both
 52 applications programs and user terminal-controlling programs to provide services
 53 of a more complex nature than the primitives provided by IEEE Std 1003.1. This
 54 interface shall be implementable on conforming IEEE Std 1003.1 systems. It shall
 55 include the following components:

- 56 1. Application program primitives to specify instructions to an
 57 implementation defined "shell" facility.
- 58 2. A standard command language for a shell that includes program execution,
 59 I/O redirection and pipelining, argument handling, variable substitution and

* An IEEE Std 1003.1 conforming *implementation* is not necessarily required to support these application programs. Implementations could be produced that are conformant only to those 1003.1 features required by the proposed 1003.2 standard, and that cannot claim full conformance to all of IEEE Std 1003.1.

- 60 expansion, and a series of control constructs similar to other high-level
61 structured programming languages.
- 62 3. A recommended command syntax for command naming and argument
63 specification.
- 64 4. Primitives to assist applications programs and the shell language in parsing
65 and interpreting command arguments.
- 66 5. Recommended environment variables for use by shell scripts and
67 application programs.
- 68 6. A minimum directory hierarchy required for the shell and applications.
- 69 7. A group of utilities that may be called from application programs for c
70 complex data manipulation and other tasks common to many applications. c
- 71 8. An optional group of utilities to be used for the software development of c
72 applications. c
- 73 9. Utilities and standards for the installation of applications. c
- 74 The following areas are outside the scope of this standard:
- 75 1. Operating system administrative commands (privileged processes, system
76 processes, daemons, etc.).
- 77 2. Commands required for the installation, configuration, or maintenance of
78 operating systems or file systems.*
- 79 3. Networking commands.
- 80 4. Terminal control or user-interface programs (visual shells, window
81 managers, command history mechanisms, etc.).
- 82 5. Graphics programs or interfaces.
- 83 6. Text formatting programs or languages.
- 84 7. Database programs or interfaces (e.g. SQL, etc.).
- 85 At the time of this printing, no published document existed. Working drafts were being
86 circulated, with a target schedule of early 1989 for balloting. c

* This is contrasted against paragraph i, above, by its orientation to installing the operating system itself, versus application programs. The exclusion of operating system installation facilities should not be interpreted to mean that the non-privileged application installation procedures *cannot* be used for installing operating system components.

87 If you are interested in participating in this effort contact the IEEE Standards Office; the
88 address is listed in the Foreword.

89 A.2.3 Verification Testing

90 This area is currently in development by IEEE Computer Society Working Group c
91 P1003.3. c

92 If you are interested in obtaining 1003.3-related documents, or in participating in this
93 effort, contact the IEEE Standards Office.

94 A.2.4 Real Time Extensions

95 A project has been approved for IEEE Computer Society Working Group P1003.4 to c
96 develop and ballot extensions to IEEE Std 1003.1 to address service interfaces needed for c
97 portable real time applications. This working group is an outgrowth of the /usr/group c
98 Technical Committee Real Time Subcommittee. At the time of publication, no draft c
99 document existed. c

100 Contact the IEEE Standards Office to participate in this effort. c

101 A.2.5 Language Standards

102 The following language standards are available from ANSI:

103	Ada	Mil Std 1815-A-1983	c
104	Basic	X3.113-1987	c
105	Cobol	X3.23-1985	c
106	Fortran	X3.9-1978	c
107	Mumps	MDC X11.1-1984	c
108	Pascal	X3.97-1983	c

109 A.2.6 Networking Standards

110 The ISO/OSI (Open System Interconnect) networking specifications are available from c
111 CBEMA or ANSI (and 802.*n* from the IEEE Standards Office): c

112	OSI Model		ISO 7498 (ANSI)	c
113	Layer 1	CSMA/CD	IEEE 802.3 (IEEE)	c
114		Token Bus	IEEE 802.4 (IEEE)	c
115		Token Ring	IEEE 802.5 (IEEE)	c
116	Layer 2	Link Layer Control	IEEE 802.2 (IEEE)	c
117			CCITT DR X.212 (CBEMA)	c
118	Layer 3	Network Layer	ISO 8348, 8473, 7777 (CBEMA)	c
119	Layer 4	Transport Layer	ISO 8072, 8073 (CBEMA)	c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

120	Layer 5	Session Layer	ISO 8326, 8327 (CBEMA)	c
121	Layer 6	Presentation Layer	ISO DP 8822, DP 8823 (CBEMA)	c
122	Layer 7	Applications Layer		c
123		CASE (Common Services)	ISO DP 8649, DP 8650 (CBEMA)	c
124		FTAM (File Transfer)	ISO DP 8571 (CBEMA)	c
125		Mail/Message	CCITT X.400 series (CBEMA)	c
126		Job Transfer	ISO DP 8831, DP 8832 (CBEMA)	c
127	Wide Area Net	Layers 1-3	CCITT X.25 (CBEMA)	c
128	A.2.7 Graphics Standards			c
129	The following graphics-related standards are available from CBEMA or ANSI:			c
130	GKS	X3.124-1985 <i>Graphical Kernel System</i> ; C language bindings are		c
131		in progress (0533-D). (ANSI)		c
132	PHIGS	X3.144-198x <i>Programmers' Hierarchical Interactive Graphics</i>		c
133		<i>System</i> ; C language bindings are in progress (0534-D). (CBEMA)		c
134	CGM	X3.122-1986 <i>Computer Graphics Metafile</i> , formerly known as		c
135		VDM, Virtual Device Metafile. (CBEMA)		c
136	X3H3.6	This working group is addressing windowing standards and		c
137		display management for graphical devices. (CBEMA)		c
138	A.2.8 Data Base Standards			
139	The following data base standards are available from ANSI:			c
140	NDL	X3.133-1986 <i>Database Language NDL</i> . (Network Databases.)		c
141	SQL	X3.135-1986 <i>Database Language SQL</i> . (Relational Databases.)		c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

- 142 **A.3 Industry Open Systems Publications**
- 143 The following publications describe recommendations formed by industry groups (as
144 opposed to a single company) about related standards efforts.
- 145 The *X/OPEN Portability Guide* is available from: c
- 146 Elsevier Science Publishers Co. Inc,
147 P.O. Box 211
148 Grand Central Station,
149 New York, NY 10163
- 150 c
- 151 **A.4 US Government Standards** c
- 152 **A.4.1 Federal Information Processing Standards (FIPS)** c
- 153 The following standards are designated by the US Government as Federal Information c
154 Processing Standards. These frequently refer back to standards listed above. c
155 Information on these can be obtained from: c
- 156 National Technical Information Service c
157 US Department of Commerce c
158 5285 Port Royal Road c
159 Springfield, VA 22161 c
160 (703) 487-4650 c
- 161 An index for FIPS standards is *NBS Publications List 58*, available as document number c
162 301-975-2816. c
- 163 **A.4.2 Trusted Systems** c
- 164 A standard for secure, or trusted, systems, the *Department of Defense Trusted Computer* c
165 *System Evaluation Criteria*, Department of Defense Standard DoD 5200.28-STD, c
166 December 1985, is available from: c
- 167 Office of Standards and Products c
168 National Computer Security Center c
169 Fort Meade, MD 20755-6000 c
170 Attn: Chief, Computer Security Standards c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

B. Rationale and Notes

- 1 This appendix summarizes the deliberations of the IEEE P1003.1 Working Group, the
2 committee charged by IEEE with devising an interface standard for a portable operating
3 system interface for computer environments, IEEE Std 1003.1. B
- 4 This appendix is derived in part from copyrighted draft documents developed under the C
5 sponsorship of /usr/group*, as part of an ongoing program of that association to support A
6 the IEEE 1003 standards program efforts. A
- 7 The appendix is being published along with the standard to assist in the process of
8 review. It contains historical information concerning the contents of the standard and
9 why features were included or discarded by the Working Group. It also contains notes of
10 interest to application programmers on recommended programming practices,
11 emphasizing the consequences of some aspects of the standard that may not be
12 immediately apparent.

* Copyright © 1987 by /usr/group. Reprint rights granted to the IEEE for this appendix.

/usr/group is a registered trademark of /usr/group, the International Network of UNIX System Users.

13 B.1 Introduction

14 The IEEE Std 1003.1 is based on the UNIX operating system developed by AT&T Bell
15 Laboratories, and derives from efforts of the Standards Committee of /usr/group, an
16 association of individuals, corporations, and institutions with an interest in the UNIX A
17 system that has long worked toward the development of independent industry-driven
18 standards. The IEEE P1003 Working Group represents a cross-section of the UNIX B
19 system community: it consists of over 250 members representing hardware C
20 manufacturers, vendors of operating systems and other software development tools,
21 software designers, consultants, academics, authors, applications programmers, and
22 others. In the course of its deliberations, it has reviewed related American and
23 international standards, both published and in progress. This revision includes responses
24 and rationale material related to the comments received in the trial use period.

25 Although originally coined by the IEEE to refer to IEEE Std 1003.1, the term POSIX more C
26 correctly refers to a *family* of related standards or working groups, P1003.*n*. These other C
27 activities are described in Appendix A. There are some cases where this rationale uses C
28 the term POSIX as a synonym for IEEE Std 1003.1. This incorrect usage is maintained C
29 for purposes of readability only. The body of the standard does not use the term POSIX C
30 in this way. C

31 As explained in the Foreword, the term POSIX is expected to be pronounced *pahz-icks*, as C
32 in *positive*, not *poh-six*, or other variations. The P1003 Working Group has published C
33 the pronunciation of its term in an attempt to promulgate a standardized way of referring C
34 to a standard operating system interface. C

35 The intended audience for this standard is all persons concerned with an industry-wide A
36 standard operating system based on the UNIX system. This includes at least four groups
37 of people:

- 38 1. persons buying hardware and software systems;
- 39 2. persons managing companies that are deciding on future corporate
40 computing directions;
- 41 3. persons implementing operating systems, and especially;
- 42 4. persons developing applications where portability is an objective. C

43 **B.1.1 Scope**

44 This Rationale focuses primarily on additions, clarifications, and changes made to the
45 UNIX system as described in the **Base Documents §B.1.3** from which the standard was
46 derived. It is *not* a rationale for the UNIX system as a whole, since the Working Group B
47 was charged with codifying existing practice, not designing a new operating system. No B
48 attempt is made in this Rationale to defend the pre-existing structure of UNIX systems. It
49 is primarily deviations from existing practice, as codified in the Base Documents, that are
50 explained or justified here.

51 The Rationale discusses some UNIX system features that were *not* adopted into the B
52 standard. Many of these are features that are popular in some UNIX system
53 implementations, so that a user of those implementations might question why they do not
54 appear in the standard.

55 There are choices allowed by the standard for some details of the interface specification; A
56 some of these are specifiable option subsets of the standard. See **Portability A**
57 **Specifications §B.2.10**. See also **Specific Derivations §B.1.3.3**. A

58 The standard is not a tutorial on the use of the specified interface, nor is this Rationale.
59 However, the Rationale includes some references to well-regarded historical books on
60 the UNIX System in **Historical Implementations §B.11.2**.

61 **B.1.2 Purpose**

62 Several principles guided the Working Group's decisions.

63 **B.1.2.1 Application Oriented**

64 The basic goal of the Working Group was to promote portability of application programs
65 across UNIX system environments by developing a clear, consistent, and unambiguous
66 standard for the interface specification of a portable operating system based on the UNIX
67 system documentation. This standard codifies the common, existing definition of the
68 UNIX system. There was no attempt to define a new system interface.

69 **B.1.2.2 Interface, Not Implementation**

70 The standard defines an interface, not an implementation. No distinction is made
71 between library functions and system calls: both are referred to as functions. No details
72 of the implementation of any function are given (although historical practice is
73 sometimes indicated in the Rationale). Symbolic names are given for constants (such as
74 signals and error numbers) rather than numbers.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

75 B.1.2.3 Source, Not Object, Portability

76 The standard has been written so that a program written and translated for execution on
 77 one conforming implementation may also be translated for execution on another
 78 conforming implementation. The standard does not guarantee that executable (object)
 79 code will execute under a different conforming implementation than that for which it was
 80 translated, even if the underlying hardware is identical. The Working Group has,
 81 however, attempted to put few impediments in the way of binary compatibility, and some
 82 remarks are found in this Rationale. See Requirements §B.2.2.1.1 and Configurable
 83 System Variables §B.4.8.

C
C

84 B.1.2.4 The C Language and X3J11

85 The standard is written in terms of the standard C language as specified in the
 86 *ANSI/X3.159-198x Programming Language C Standard* that the X3J11 Working Group
 87 produced. See Conformance §2.2. Guidelines used in negotiations between the two
 88 Working Groups are discussed below in C Language, X3J11, and P1003.1 §B.1.4.

A
A

89 B.1.2.5 No Super-User, No System Administration

90 There was no intention to specify all aspects of an operating system. System
 91 administration facilities and functions are excluded from the standard, and functions
 92 usable only by the super-user have not been included. This Rationale notes several such
 93 instances. Still, an implementation of the standard interface may also implement features
 94 not in the standard: see Requirements §2.2.1.1. The standard is also not concerned with
 95 hardware constraints or system maintenance.

A
A

96 B.1.2.6 Minimal Interface, Minimally Defined

97 In keeping with the historical design principles of the UNIX system, the standard is as
 98 minimal as possible. For example, it usually specifies only one set of functions to
 99 implement a capability. Exceptions were made in some cases where long tradition and
 100 many existing applications included certain functions, such as *creat()* §5.3.2. In such
 101 cases, as throughout the standard, redundant definitions were avoided: *creat()* §5.3.2
 102 defined as a special case of *open()* §5.3.1. Redundant functions or implementations with
 103 less tradition were excluded. For example, *seekdir()* §B.5.1.2 and *tellidir()* §B.5.1.2
 104 were not included in Directory Operations §5.1.2.

105 B.1.2.7 Broadly Implementable

106 The Working Group has endeavored to make all specified functions implementable
 107 across a wide range of existing and potential systems, including:

- 108 • All of the current major systems that are ultimately derived from AT&T code
 109 (Version 7 or later).
- 110 • Compatible systems that are not derived from AT&T code.
- 111 • Emulations hosted on entirely different operating systems.
- 112 • Networked systems.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

113 • Distributed systems.

114 • Systems running on a broad range of hardware.

115 No direct references to this goal appear in the standard, but some results of it are
116 mentioned in this Rationale.

117 **B.1.2.8 Minimal Changes to Historical Implementations**

118 There is no known historical implementation §B.2.3 that will not have to change in
119 some area to conform to the standard, and in a few areas the standard does not exactly
120 match any existing system interface (for example, see O_NONBLOCK §B.6).
121 Nonetheless, there is a set of functions, types, definitions, and concepts that form an
122 interface that is common to most historical implementations. The standard specifies that
123 common interface and extends it in areas where there has historically been no consensus,
124 preferably

125 1. by standardizing an interface like one in an historical implementation, e.g.,
126 Directories §5.1, or

127 2. by specifying an interface that is readily implementable in terms of, and backwards A
128 compatible with, existing implementations, such as TAR §10.1, or

129 3. by specifying an interface that, when added to a historical implementation, will not
130 conflict with it, like O_NONBLOCK §B.6.

131 Required changes to historical implementations have been kept as few as possible, but
132 they do exist, and this Rationale points out some of them.

133 The standard is specifically not a codification of a particular vendor's product. It is like
134 the UNIX system, but it is not identical to it. The word UNIX is not used in the standard
135 proper both for that reason, and because it is a trademark of a particular vendor.

136 **B.1.2.9 Minimal Changes to Existing Application Code**

137 The Working Group wished to make less work for application developers, not more.
138 However, because every known historical implementation will have to change at least
139 slightly to conform, some applications will have to change. This Rationale points out the
140 major places where the standard implies such changes.

141 **B.1.2.10 IEEE Consensus Process**

142 The IEEE consensus process was used in deliberations. There are several levels of A
143 participation: A

144 • **Correspondents.** A

145 Those interested in following the development of the standard could subscribe A
146 to a mailing list to which copies of drafts, working documents, and related A
147 material were sent. Also, anyone (including individuals, companies, A
148 government agencies, or other organizations) could send comments (or RFCs, A
149 Proposals, or Notes) to the Working Group. A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 150 • **Working Group.** A
- 151 This was the group responsible for producing the standard document. It met A
- 152 four times a year and produced many drafts. It also produced the Trial Use A
- 153 and Full Use Standards, and was responsible for resolving balloting objections A
- 154 to them. The Working Group was composed of individuals, even though A
- 155 many of them worked for companies with interests in the field. A
- 156 • **Balloting Group.** A
- 157 This group voted on the proposed standards in the manner detailed in the next A
- 158 subsection. The Balloting Group, like the Working Group, was composed of A
- 159 individuals. Most of the people on the Working Group also were in the A
- 160 Balloting Group, although the latter included many others, as well. A
- 161 • **Institutional Representatives.** A
- 162 Exceptions to the individual composition of the Balloting Group were the A
- 163 Institutional Representatives, who represented related standards bodies or A
- 164 professional organizations (in this case, USENIX, /usr/group, and X/OPEN). A
- 165 These Institutional Representatives also served on the Working Group, but A
- 166 participated there as individuals. A
- 167 Decisions of the Working Group were not made by vote, not even of a large majority. A
- 168 Decisions were made by consensus, which required that each individual believe that A
- 169 • their point of view had been heard A
- 170 • their point of view had been understood A
- 171 • other individuals' points of view were adequately understood A
- 172 • there was general consensus. A
- 173 A common way of moving discussion along was to ask if anyone would ballot "no" on a A
- 174 particular issue. A
- 175 **B.1.2.11 IEEE Balloting Process**
- 176 The IEEE balloting process is used to attain the ANSI requirement for a consensus C
- 177 acceptance of a document as a standard. C
- 178 Balloting in IEEE is done by individuals who are members of IEEE or affiliated with the C
- 179 IEEE Computer Society. They are given thirty days in which to return the ballots, and C
- 180 75% of those in the balloting group must return ballots. C
- 181 Ballots from non-IEEE members are also included in the process, with comments and C
- 182 objections treated the same as those from members. However, non-IEEE members are C
- 183 not included in the percentages of returns required or the affirmative percentage required C
- 184 for approval. Possible ballot responses [excluding abstentions] are: C
- 185 • yes without comments C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

- 186 • **yes with comments** c
 187 The comments indicate areas that should be evaluated, but are not significant c
 188 enough to warrant a negative ballot. c
- 189 • **no with objections** c
 190 A negative ballot must include specific objections and recommendations on c
 191 how to resolve the objections. These objections indicate areas that must be c
 192 fixed to resolve the negative ballot. c
- 193 At least 75% of those balloting [not abstaining] must provide an affirmative response. c
 194 Each objection, and many of the comments, are translated into proposed changes; and c
 195 any outstanding objections, along with the rationale for not making the changes to c
 196 accommodate these objections, are fed back to the balloting group. c
- 197 Members of the balloting group are given ten days to change their ballots, with similar c
 198 options as above; however, objections are limited to the proposed changes and/or failure c
 199 to resolve key objections. It is possible for the number of negative responses to increase c
 200 if a proposed change is objectionable, or if a significant objection has not been addressed. c
- 201 In general, the balloting process moves fairly quickly towards a high degree of c
 202 consensus. The final results are submitted to the IEEE Standards Board for approval, and c
 203 include the balloting percentages as well as documentation of any unresolved negative c
 204 objections. c
- 205 The Trial Use period was from April 1986 to the November 1987, when the balloting of c
 206 the revised document [Draft 12] began, and provided an additional level of industry c
 207 consensus. The high visibility of the document, as well as its widespread distribution, c
 208 provided additional feedback and information for the formulation of the current standard. c
 209 See also Specific Derivations §B.1.3.3. c
- 210 The Institutional Representatives were exceptions in several ways. c
- 211 • They are not required to be IEEE members. c
 212 • They ballot for their Institutions, not as individuals. c
 213 • Ballots of Institutional Representatives are reported separately to the IEEE c
 214 Standards Board. c
- 215 As with other ballots, any unresolved negative objections are reported with the rationale c
 216 for not incorporating the associated changes. However, the separate reporting of the c
 217 Institutional ballots tends to make any objections more visible, particularly in that c
 218 Institution's areas of expertise; consequently, any unresolved objection could be enough c
 219 to cause the document to be sent back to the balloting process for further resolution. c
 220 USENIX,balloted affirmative for the Trial Use Standard; /usr/group balloted negative, and c
 221 their unresolved issue was mandatory locking; X/OPEN did not ballot. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

222 B.1.3 Base Documents

223 The Working Group consulted a number of documents as representing features
224 appropriate for consideration for inclusion in the standard. Full bibliographic
225 information may be found in Bibliographic Notes §B.11.

226 B.1.3.1 Related Standards and Documents

- 227 • *1984 /usr/group Standard*
- 228 • *ANSI/X3.159-198x Programming Language C Standard*
- 229 • *X/OPEN Portability Guide*

230 The most direct ancestor is the *1984 /usr/group Standard*, which is considered to be
231 Draft 1 of the present standard. It, in turn, was largely derived from the programming
232 interface of System III. The *1984 /usr/group Standard* is also the principal ancestor of
233 the Library section of the C Standard.

234 The X3J11 and P1003.1 Working Groups have cooperated closely. Details of the
235 relations of the two standards they produced are listed in this Rationale in C Language,
236 X3J11, and P1003.1 §B.1.4 because the C Standard is the standard most closely related
237 to POSIX. POSIX is written in terms of the C Standard, although it is possible to have
238 POSIX without Standard C: see Conformance §B.2.2.

239 The *X/OPEN Portability Guide* proved useful because X/OPEN had in many cases already
240 addressed the same issues as P1003.1, though often in a slightly different context.

241 The Working Group is aware of the Japanese SIGMA project, which includes as a goal a
242 common operating system interface specification, and there has been a representative of
243 SIGMA at most recent P1003.1 Working Group meetings.

244 B.1.3.2 Historical Implementations

245 These include (with colloquial names in parentheses):

- 246 • *UNIX Time-Sharing System: UNIX Programmer's Manual, Seventh Edition*
247 *(Version 7)*
- 248 • *UNIX System III Programmer's Manual*
- 249 • *AT&T System V Interface Definition (SVID), Issue 2, Volumes 1-3*
- 250 • *4.3 Berkeley Software Distribution, Virtual VAX-11 Version (4.3BSD)*
251 *Manuals*

252 The UNIX system has changed more since the *1984 /usr/group Standard* was written than
253 has the C language, and there are more variants of the former. Because of this, the
254 present standard has been radically reorganized and reformatted since the first draft and
255 has had many changes in content. Thus there is no single Base Document to provide
256 context for all discussions in this Rationale, which instead discusses aspects of Version 7,
257 System III, System V, and 4.3BSD that were included in this standard or that were

- 258 considered in choosing what was included.
- 259 Occasional mentions are made of Version 8 and Version 9, which are successors of
 260 Version 7, the Bell Laboratories research system. The context is usually related to the
 261 *streams* inter-process communication mechanism, which is not in this standard but which
 262 has influenced discussions about inter-process communication mechanisms.
- 263 Although 4.2BSD was the current Berkeley Software Distribution when most of the work
 264 on the standard was done, this Rationale refers to 4.3BSD instead (in most places)
 265 because the differences between the two versions are almost entirely in performance, the
 266 few programming interface differences are mostly outside the scope of this standard, and
 267 the 4.3BSD manuals actually describe 4.2BSD better than the 4.2BSD manuals do.
- 268 The System V manuals are never referenced because the *SVID* is more definitive.
- 269 Much of the standard is closer to the *SVID* than to any other document, and there is an
 270 appendix that compares the two directly.
- 271 Parts of documentation of many other related systems were considered in deliberations
 272 on various aspects of the standard. As those were too numerous to list all of them, none
 273 of them will be mentioned by name.
- 274 **B.1.3.3 Specific Derivations**
- 275 Some areas of the standard are clearly derived from facilities of specific systems. Most
 276 of the major areas are listed here, together with references to the sections of the standard
 277 where they occur. For most of them, there is also more detail in the corresponding
 278 sections of the Rationale.
- 279 FIFOs
- 280 The FIFO special file §2.3 facility exists in System III, the 1984
 281 */usr/group Standard*, and System V, but not in Version 7, 4.2BSD,
 282 or 4.3BSD.
- 283 reliable signals
- 284 Signals §3.3 includes reliable signals related to the 4.3BSD model.
 285 These were introduced between the Trial Use and Full Use
 286 Standards. A
- 287 job control A
- 288 The job control §B.3.3 facility is derived from 4.3BSD and was A
 289 introduced between the Trial Use and Full Use Standards. A
- 290 saved set-user-ID (saved set-group-ID)
- 291 This optional capability, mostly in *exec* §3.1.2 and Set User and
 292 Group IDs §4.2.2, is derived from System V, and was introduced
 293 in the Trial Use Standard.
- 294 supplementary groups
- 295 A single group per process as in System V is the default, but User

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 296 Identification §4.2 (particularly *getgroups()* §4.2.3) allows
 297 multiple groups per process as in 4.3BSD as an option. This was
 298 introduced shortly before the Trial Use Standard.
- 299 *uname()*
 300 The *uname()* §4.4.1 function is derived from the 1984 *lusr/group*
 301 *Standard*, which took it from System III, and it is still in System V.
 302 It does not exist in Version 7 or 4.3BSD.
- 303 *opendir()*, *readdir()*, *rewinddir()*, *closedir()*
 304 Directory Operations §5.1 is derived from 4.2BSD and was
 305 introduced in an early draft of the standard. It was later adopted in
 306 System V Release 3.
- 307 *mkdir()*, *rmdir()*, *rename()*
 308 The three functions *mkdir()* §5.4.1, *rmdir()* §5.5.2, and *rename()*
 309 §5.5.3 are derived from 4.2BSD. Except for *rename()*, these c
 310 functions now also appear in System V Release 3. c
- 311 *termios*
 312 Device- and Class-Specific Functions §7, while closer to
 313 System V than to 4.3BSD, does not correspond to any existing
 314 system because none was found adequate when considerations
 315 such as international character sets, fast interfaces, and networks
 316 were taken into account. The final interface specification was
 317 introduced shortly before the Full Use Standard.
- 318 archive format
 319 The Extended tar Format §D.1 is derived from the *tar* programs c
 320 used in Version 7 and 4.3BSD, and provided with System V. The c
 321 precise format in the Full Use Standard has evolved incrementally c
 322 from that in earlier drafts of POSIX. c
- 323 B.1.3.4 Working Documents
 324 The model for the present Rationale was the Rationale prepared by the X3J11 Working A
 325 Group to accompany the *ANSI/X3.159-198x Programming Language C Standard*: A
- 326 • *X3J11/86-152, October 1, 1986 "Rationale for Draft Proposed American A*
 327 *National Standard for Information Systems—Programming Language C"* A
- 328 Its influence may be seen most clearly in C Language, X3J11, and P1003.1 §B.1.4, but A
 329 it also is present in more subtle ways throughout. A
- 330 References to programs, functions, or facilities of systems described by the Base
 331 Documents (such as the System V *cpio* utility program) have been freely included in
 332 this Rationale where relevant, even though they would be inappropriate in the standard
 333 itself. References to programs, functions, or facilities not described by the base
 334 documents or to companies not directly associated with them have been excluded where

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

335 possible. Exceptions have been made where facilities were derived from systems not
 336 described by the base documents, and where the word “may” is used to describe an
 337 option that permits behavior of such a system. A
 A
 A

338 B B

339 B.1.4 C Language, X3J11, and P1003.1

340 Some C language functions and definitions were handled by P1003.1, but most by X3J11.
 341 The most general guideline was that P1003.1 retained responsibility for operating-system
 342 specific functions, while X3J11 defined C library functions. See also C Language
 343 Definitions §B.2.8 and C Language Library §B.8. .

344 There are several areas in which the two standards differ philosophically:

345 • **Function parameter type lists.**
 346 These appear in the C Standard and specify the types of the arguments and
 347 return values of functions in external references to them. POSIX does not
 348 include them, except in a few places to indicate variable number of
 349 arguments, e.g., File Control §B.6.5.2. Function parameter type lists were A
 350 not used because the Working Group was aware that some vendors would A
 351 wish to implement POSIX in terms of a binding to an historical variant of the A
 352 C language instead of to the *ANSI/X3.159-198x Programming Language C* A
 353 *Standard*, since compilers for the latter would initially not be widespread. A
 354 Since the C Standard does not require the use of function parameter type lists, A
 355 the function definitions used in POSIX are nonetheless specified in terms of A
 356 Standard C. POSIX implementors whose C implementations support ANSI- C
 357 style function prototypes should consider using them for declarations in C
 358 POSIX. (Note that some code with improper declarations may have problems C
 359 if this is done.) See also *signal()* §B.3.3.3. C

360 • **Single vs. multiple processes.**
 361 The C Standard specifies a language that can be used on single-process
 362 operating systems and as a freestanding base for the implementation of
 363 operating systems or other stand-alone programs. But the POSIX interface is
 364 that of a multi-process timesharing system. Thus POSIX has to take multiple
 365 processes into account in places where the C Standard does not mention
 366 processes at all, such as *kill()* §3.3.2. See also Requirements §B.2.2.1.1.

367 • **Single vs. multiple operating system environments.**
 368 The C Standard specifies a language that may be useful on more than one
 369 operating system, and thus has means of tailoring itself to the particular
 370 current environment. POSIX is an operating system interface specification,
 371 and thus by definition is only concerned with one operating system
 372 environment, even though it has been carefully written to be broadly
 373 implementable §B.1.2.7 in terms of various underlying operating systems.
 374 See also Requirements §B.2.2.1.1.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 375 • **Translation vs. execution environment.**
 376 POSIX is primarily concerned with the Standard C execution environment, A
 377 leaving the translation environment to the C Standard. See also A
 378 Requirements §B.2.2.1.1. A
- 379 • **Hosted vs. freestanding implementations.** A
- 380 All POSIX implementations are hosted in the sense of the C Standard. See C
 381 also the remarks on conformance in the Foreword. C
- 382 • **Text vs. binary file modes.**
 383 X3J11 defines “text” and “binary” modes for a file. But the POSIX interface
 384 and historical implementations related to it make no such distinction, and all
 385 functions defined by P1003.1 treat files as if these modes are identical. (It is C
 386 important not to say that POSIX files are either “text” or “binary.”) X3J11 A
 387 wrote their definitions so that this interpretation is possible. In particular, A
 388 “text” mode files are not required to end with a line separator, which also A
 389 means that they are not required to include a line separator at all. A
- 390 And there is a basic difference in approach between the X3J11 Rationale and the P1003.1 A
 391 Rationale. The X3J11 Rationale addresses almost all changes as differences from the A
 392 Base Documents of the C Standard, usually either Kernighan and Ritchie or the 1984 A
 393 *usr/group Standard*. The present Rationale cannot do that, since there are many more A
 394 variants of (and Base Documents for) the operating system interface than for the C A
 395 language. The most noticeable aspect of this difference is that X3J11 marks QUIET A
 396 CHANGES from the Base Documents in its Rationale. The POSIX Rationale cannot A
 397 include such markings, since a quiet change from one historical implementation may A
 398 correspond exactly to another historical implementation, and may be very noticeable to A
 399 an application written for yet another. A
- 400 **B.1.4.1 Solely by P1003.1.**
 401 These return parameters from the operating system environment: *cuserid()* §4.2.4,
 402 *ctermid()* §4.7.1, *ttyname()* §4.7.2, and *isatty()* §4.7.2.
- 403 The functions *fileno()* §8.2.1 and *fdopen()* §8.2.2, map between C Language stream
 404 pointers and POSIX file descriptors.
- 405 **B.1.4.2 Solely by X3J11.**
 406
- 407 There are many functions that are useful with the operating system interface and are
 408 required for conformance with the present standard, but that are properly part of the
 409 C Language. These are listed in Referenced C Language Routines §8.1, which also
 410 notes which functions are defined by both P1003.1 and X3J11. Certain terms defined by
 411 X3J11 are incorporated by P1003.1 in C Language Definitions §2.8.
- 412 Some routines were considered too specialized by the P1003.1 Working Group to be c
 413 included in the standard. These include *bsearch()* and *qsort()*.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

414 **B.1.4.3 By Neither P1003.1 nor X3J11.**

415 Some functions were considered of marginal utility and problematical when international
416 character sets were considered: *_toupper()*, *_tolower()*, *toascii()*, and *isascii()*.

417 Though *malloc()* §8.1 and *free()* §8.1 are in the C Standard and are required by
418 Referenced C Language Routines §8.1 of the present standard, neither *brk()* §B.1.4.3 A
419 nor *sbrk()* §B.1.4.3 occur in either standard (although they were in the 1984 *lusr/group* A
420 *Standard*), because this standard is designed to provide the basic set of functions required
421 to write a Conforming Application; the underlying implementation of *malloc()* or *free()*
422 is not an appropriate concern for the standard.

423 **B.1.4.4 Base by P1003.1, Additions by X3J11.**

424 Since the C Standard does not depend on POSIX in any way, there are no items in this A
425 category. A

426 **B.1.4.5 Base by X3J11, Additions by P1003.1.**

427 X3J11 has to define *errno* if only because examining that variable is the only way to tell
428 when some mathematics routines fail. But P1003.1 uses it more extensively, and adds c
429 some semantics to it in Error Numbers §2.5, which also defines some values for it. c

430 Many numerical limits used by X3J11 were incorporated by P1003.1 in Numerical
431 Limits §2.9, and some new ones are added, all to be found in the header *<limits.h>*.

432 The semantics of arguments to *main()* §3.1.2 are only defined in POSIX.

433 The POSIX definition of *signal()* §8.3.2 further specifies the C definition, and the entire c
434 mechanism of *signals* §3.3 is much more elaborate. c

435 The function *time()* §4.5.1 is used by X3J11, but POSIX further specifies the time value. c

436 The function *getenv()* §4.6.1 is referenced in Environment Description §2.7 and *exec*
437 §3.1.2 and is also defined by X3J11.

438 The function *rename()* §5.5.3 is extended to further specify its behavior when the new
439 filename already exists or either argument refers to a directory.

440 **B.1.4.6 Related Functions by Both.**

441 The X3J11 definition of compliance and the P1003.1 definition of Conformance §2.2 are
442 similar, although the latter notes certain potential hardware limitations.

443 P1003.1 defined a portable filename character set in General Terms §2.3, that is like the
444 X3J11 identifier character set. However, P1003.1 did not allow upper- and lowercase
445 characters to be considered equivalent. See filename portability §2.4.

446 The type *clock_t* §2.6 appears in both standards. See *Time* §B.4.5.

447 The *exit()* function is defined only by X3J11, because it refers to closing streams, and
448 that subject, as well as *fclose()* itself, is defined almost entirely by X3J11. But P1003.1
449 defined *_exit()* §3.2.2, which also adds semantics to *exit()*. This also allows POSIX to c
450 ignore the X3J11 *atexit()* function. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

451 P1003.1 defined *kill()* §3.3.2, while X3J11 defined *raise()*, which is similar except that it
 452 does not have a process ID argument, since the language defined by X3J11 does not
 453 incorporate the idea of multiple processes.

454 The new functions *sigsetjmp()* §8.3.1 and *siglongjmp()* §8.3.1 were added to provide
 455 similar functions to X3J11 *setjmp()* and *longjmp()* that additionally save and restore
 456 signal state. Requiring *setjmp()* and *longjmp()* to do this would have conflicted with the
 457 X3J11 definitions.

458 B.1.5 Organization

459 B.1.5.1 Organization of the Standard

460 See the Foreword.

461 It was decided very early that the traditional organization by manual section, as used in
 462 the 1984 *usr/group Standard*, would be confusing in an IEEE standard. That
 463 organization assumed some background that was not relevant to the purpose of the
 464 standard. It also made an implementation-oriented distinction between system calls and
 465 library routines, which were in separate sections.

466 Two sections, Scope §1 and Definitions §2, have been prepended because they are
 467 traditional in IEEE standards. A Foreword was prepended for the same reason, even
 468 though it is not part of the standard proper. The name POSIX, suggested by Richard
 469 Stallman, was adopted during the printing of the Trial Use Standard.

470 Although appendices were used in the Trial Use Standard to contain proposals for
 471 examination by the Balloting Group and the general public, the Full Use Standard has no
 472 proposal appendices, because the text of the standard proper must be complete. The
 473 Appendices of the Full Use Standard discuss either related standards or the Full Use
 474 Standard itself. *Editor's Note: Appendices D and E are an exception to the preceding*
 475 *two sentences. They will not appear in the Full Use Standard after it is approved, being*
 476 *included only to expedite the balloting process.* The Full Use Standard contains some
 477 new material that was not in the Trial Use Standard, mostly that which was added to
 478 meet balloting objections. The most obvious examples are the addition of reliable signal
 479 considerations to Signals §3.3 (including the addition of Non-Local Jumps §8.3.1) and
 480 the resolution of Device- and Class-Specific Functions §7. See also Specific
 481 Derivations §B.1.3.3.

482 Because there were too many notes interpolated in the text of the Trial Use Standard
 483 (which were nonetheless not part of the standard), and because there were still not
 484 enough to explain why the Working Group had made many difficult decisions, the
 485 Working Group decided to add a Rationale and Notes Appendix, modeled after the one
 486 the X3J11 Working Group was producing for the C Standard. Most of the notes formerly
 487 in the main body of the draft were moved to the Rationale appendix, although some were
 488 deleted and others were incorporated into the text of the standard proper.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

489 B.1.5.2 Organization of this Appendix

490 Just as the standard proper excludes all examples, footnotes, references, and appendices,
 491 this Rationale is also not part of the standard. The POSIX interface is defined by the
 492 standard alone. If any part of this Rationale is not in accord with that definition, the IEEE
 493 Standards Office should be so informed. In the meantime, conflicts between this
 494 Rationale and the standard are always resolved in favor of the body of the standard. A
 C
 C
 C

495 All sections of this appendix after this first major section, **Introduction §B.1**, follow the
 496 exact structure of the standard, and aspects of a given section of the standard are
 497 considered in the corresponding section of the Rationale. Where a given discussion
 498 touches on several areas, attempts have been made to include cross-references within the
 499 text.

500 References to the standard are in the same format as references within the standard to
 501 parts of itself, for example: **Definitions §2.0**. References to this Rationale are given as
 502 references to Appendix B of the standard, that is, the section numbers always begin with
 503 "B." as in **Definitions §B.2.0**. Where a reference both to part of the standard and to a
 504 related note in the Rationale would be appropriate only the latter is given, because all
 505 parts of the Rationale implicitly refer to the corresponding parts of the standard.

506 B.1.5.3 Typographical Conventions

507 Words in all capital letters (including error numbers, environment variables, and limits)
 508 are one point size smaller than regular text, e.g.: POSIX.

<u>Reference</u>	<u>Example</u>	
Command Name	cpio	B
Data Types	<i>long</i>	B
Defined Terms	file	B
Environment Variables	PATH	B
Error Numbers	[EINTR]	B
Function Arguments	<i>arg0</i>	B
Functions	<i>open()</i>	B
Global Externals	<i>errno</i>	B
Header Files	<sys/stat.h>	B
Limits	{OPEN_MAX}	B
Section References	Process Termination §3.2	B
Symbolic Constants	{_POSIX_V_DISABLE}	B

523 Defined names that are normally in lowercase, particularly function names, are never
 524 used at the beginning of a sentence or anywhere else that normal English usage would
 525 require them to be capitalized.

526 The above typographical conventions apply to both the standard and to this Rationale. A
 527 There are also some conventions peculiar to the Rationale, regarding standards for the A
 528 operating system interface and for the C language. These are used frequently in C A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

529 Language, X3J11, and P1003.1 §B.1.4:

<u>Topic</u>	<u>Operating System Interface</u>	<u>C Programming Language</u>	
Working Group standard	P1003.1 IEEE Std 1003.1	X3J11 <i>ANSI/X3.159-198x Programming Language C Standard</i>	A A A
short name	POSIX	C Standard	A
Rationale	Appendix B	<i>Rationale for American National Standard for Information Systems—Programming Language C</i>	C C
short name	this Rationale	X3J11 Rationale	C A

539 The name POSIX is usually used for the IEEE Std 1003.1 instead of the name 1003.1,
540 because the latter is too easily confused with the name of the Working Group, P1003.1. A
A

541 “Standard C” will eventually come to mean “ISO C,” but currently refers to the
542 *ANSI/X3.159-198x Programming Language C Standard* produced by the X3J11 Working
543 Group. A
A

544 B

545 B.2 Definitions and General Requirements

546 B.2.1 Terminology

547 The meanings specified in the standard for the words “shall,” “should,” and “may” are
548 mandated by IEEE. A
A

549 In this Rationale, the words “shall,” “should,” and “may” are sometimes used to
550 illustrate similar usages in the standard. However, the Rationale itself does not specify
551 anything regarding implementations or applications; see Organization of this Appendix
552 §B.1.5.2. A
A

553 implementation defined

554 This definition is analogous to that of the C Standard, and, together with undefined
555 and unspecified, provides a range of specification of freedom allowed to the
556 interface implementor. A
A
A

557 may

558 The use of “may” has been limited as much as possible, due both to confusion
559 stemming from its ordinary English meaning, and to objections regarding the
560 desirability of having as few options as possible and those as clearly specified as
561 possible. A
A
C

562 shall

563 Declarative sentences are sometimes used in the standard as if they included the
564 word “shall,” and facilities thus specified are no less required.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

565 **should**

566 In this standard, the word “should” does not usually apply to the implementation,
 567 but rather to the application. Thus the important words regarding implementations
 568 are “shall,” which indicates requirements, and “may,” which indicates options.

569 **undefined**

570 See implementation defined.

571 **unspecified**

572 See implementation defined.

573 **B.2.2 Conformance**

574 The definition of conforming implementations §2.2.1 allows application developers to
 575 know what they can depend on in an implementation.

576 There is no definition of a strictly conforming implementation; that would be an C
 577 implementation that provides *only* those facilities specified by the standard with no C
 578 extensions whatsoever. This is because no actual operating system implementation can
 579 exist without system administration and initialization facilities that are beyond the scope
 580 of the present standard.

581 The definitions of a Conforming Application Using Extensions §B.2.2.2 and of a A
 582 Strictly Conforming Application §B.2.2.3 guide users or adaptors of applications in A
 583 determining on which implementations an application will run and how much adaptation A
 584 would be required to make it run on others. These two definitions are modeled after A
 585 related ones in the C Standard. A

586 These three conformance definitions are descended from those of conforming A
 587 implementation, conforming application, and conforming portable application, A
 588 respectively, of the Trial Use Standard, but were changed to clarify A

589 1. extensions, options, and limits, A

590 2. relations among the three terms, and A

591 3. relations between POSIX and the C Standard. A

592 **B.2.2.1 Implementation Conformance**593 **B.2.2.1.1 Requirements**

594 The word “support” is used rather than “provide” in order to allow an implementation
 595 that has no resident software development facilities but which supports the execution of a
 596 Strictly Conforming Application to be a conforming implementation. See also A
 597 Translation vs. Execution Environment §B.1.4. A

598 **B.2.2.1.2 Documentation**

599 The conforming documentation should use the same numbering scheme as this standard C
 600 for purposes of cross referencing. (This also eliminates the need for a definitive “laundry C
 601 list.”) C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

602 This proposal is consistent with and supplements the verification test suite developed by c
 603 the P1003.3 working group. All options that an implementation chooses should be listed c
 604 in `<limits.h>` and `<unistd.h>`. c

605 **Hardware Failures:** Many systems incorporate buffering facilities, maintaining updated c
 606 data in volatile storage and transferring such updates to nonvolatile storage c
 607 asynchronously. Various exception conditions, such as a power failure or a system crash, c
 608 can cause this data to be lost. The data may be associated with a file that is still open, c
 609 with one that has been closed, with a directory, or with any other internal system data c
 610 structures associated with permanent storage. This data can be lost, in whole or part, so c
 611 that only careful inspection of file contents could determine that an update did not occur. c

612 Also, interrelated file activities, where multiple files and/or directories are updated, or c
 613 where space is allocated or released in the file system structures, can leave c
 614 inconsistencies in the relationship between data in the various files and directories, or in c
 615 the file system itself. Such inconsistencies can break applications that expect updates to c
 616 occur in a specific sequence, so that updates in one place correspond with related updates c
 617 in another place. c

618 For example, if a user creates a file, places information in the file, and then records this c
 619 action in another file, a system or power failure at this point followed by restart may c
 620 result in a state in which the record of the action is permanently recorded, but the file c
 621 created (or some of its information) has been lost. The consequences of this to the user c
 622 may be arbitrarily bad. For such a user on a system, the only safe action may be to c
 623 require the system administrator to have a policy that requires, after any system or power c
 624 failure, that the entire file system must be restored from the most recent backup copy c
 625 (causing all intervening work to be lost). c

626 The characteristics of each implementation will vary in this respect, and may or may not c
 627 meet the requirements of a given application or user. Enforcement of such requirements c
 628 is beyond the scope of this standard. It is up to the purchaser to determine what facilities c
 629 are provided in an implementation that affect the exposure to possible data or sequence c
 630 loss, and also what underlying implementation techniques and/or facilities are provided c
 631 that reduce or limit such loss, or its consequences. c

632 B.2.2.2 Application Conformance

633 B.2.2.2.1 Strictly Conforming Application

634 This definition is analogous to that of a Standard C conforming program.

635 The major difference between a Strictly Conforming Application and a Standard C
 636 strictly conforming program is that the latter is not allowed to use features of POSIX
 637 that are not in the C Standard.

638 Due to possible requirement for configuration or implementation characteristics in excess
 639 of the specifications in `<limits.h>` §2.9 or related to the hardware (such as array size or A
 640 file space), not every Conforming Application Using Extensions will run on every

641	conforming implementation.	B
642	B.2.2.2.2 Conforming Application	B
643	B.2.2.2.3 Conforming Application Using Extensions	B
644	B.2.2.3 Language Conformance	B
645	B.2.2.3.1 C Language Binding	B
646	The information concerning the use of library functions was adapted from a description	B
647	in the C Standard. Here is an example of how an application program can protect itself	B
648	from library functions that may or may not be macros, rather than true functions:	B
649	The <i>atoi()</i> function may be used in any of several ways:	B
650	1. by use of its associated header (possibly generating a macro expansion)	B
651	<pre>#include <stdlib.h></pre>	B
652	<pre>/* ... */</pre>	B
653	<pre>i = atoi(str);</pre>	B
654	2. by use of its associated header (assuredly generating a true function call)	B
655	<pre>#include <stdlib.h></pre>	B
656	<pre>#undef atoi</pre>	B
657	<pre>/* ... */</pre>	B
658	<pre>i = atoi(str);</pre>	B
659	or	B
660	<pre>#include <stdlib.h></pre>	B
661	<pre>/* ... */</pre>	B
662	<pre>i = (atoi) (str);</pre>	B
663		C
664	3. by explicit declaration	
665	<pre>extern int atoi (const char *);</pre>	
666	<pre>/* ... */</pre>	
667	<pre>i = atoi(str);</pre>	
668	4. by implicit declaration	
669	<pre>/* ... */</pre>	
670	<pre>i = atoi(str);</pre>	
671	(Assuming no function prototype is in scope. This is not allowed by X3J11	c
672	for functions with variable arguments; furthermore, parameter type	c
673	conversion “widening” is subject to different rules in this case.)	c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

674 Note that the C Standard reserves names starting with `'_'` for the c
 675 compiler. Therefore, the compiler could, for example, implement an
 676 intrinsic, built-in function `_asm_builtin_atoi()`, which it recognized and
 677 expanded into inline assembly code. Then, in `<stdlib.h>`, there would be
 678 the following:

```
679         #define atoi(X) _asm_builtin_atoi(X)
```

680 The user's "normal" call to `atoi()` would then be expanded inline, but the
 681 implementor would also be required to provide a callable function named c
 682 `atoi()` for use when the application requires it; for example, if its address is c
 683 to be stored in a function pointer variable. c

684 B.2.3 General Terms

685 Many of these definitions are necessarily circular, and some of the terms (such as
 686 process) are variants of basic computing science terms that are notoriously hard to
 687 define. Some are defined by context in the prose topic descriptions of General Concepts
 688 §2.4, but most appear in the alphabetical glossary format of General Terms §2.3. All
 689 technical terms not explicitly defined have definitions in the *IEEE Dictionary*. See B
 690 Bibliographic Notes §B.11.1. B

691 Some definitions must allow extension to cover terms or facilities that are not explicitly
 692 mentioned in the standard. For example, the definition of file must permit interpretation
 693 to include streams, as found in Version 8. The use of abstract intermediate terms (such
 694 as object in place or in addition to file) has mostly been avoided in favor of careful
 695 definition of more traditional terms. c

696 Some terms in the following list of notes do not appear in the standard; these are marked
 697 with a prepended asterisk (*). Many of them have been specifically excluded from the
 698 standard because they concern system administration, implementation, or other issues
 699 that are not specific to the programming interface. Those are marked with a reason, such
 700 as "implementation defined."

701 appropriate privileges B

702 One of the fundamental security problems with UNIX systems has been that the B
 703 privilege mechanism is monolithic—a user has either no privileges or *all* B
 704 privileges. Thus, a successful "trojan horse" attack on a privileged process B
 705 defeats all security provisions. Therefore, the standard allows more granular B
 706 privilege mechanisms to be defined. For many existing implementations of the B
 707 UNIX system, the presence of the term appropriate privileges in this standard B
 708 may be understood as a synonym for super-user (UID 0). However, future c
 709 systems will undoubtedly emerge where this is not the case and each discrete c
 710 controllable action will have appropriate privileges associated with it. c

711 controlling terminal A

712 The question of which of possibly several special files referring to the terminal is A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 713 meant is not addressed in the standard. A
- 714 ***cooperating implementation** C
- 715 This refers to a POSIX implementation that is done in combination with some other C
 716 set of system specifications. This might be as simple as supporting a POSIX C
 717 environment concurrently with some specific version of AT&T's UNIX Operating C
 718 System, or as complex as providing the POSIX environment with some different C
 719 vendor's products, such as MS/DOS from Microsoft, VMS from Digital Equipment C
 720 Company, etc. A cooperating environment would fall somewhere on the gray C
 721 scale from hosted implementations to native, depending on the degree of POSIX C
 722 components that are serviced directly versus those that are converted to correspond C
 723 with one of the other system's implementations. (Note that the POSIX facilities C
 724 might be native, and the other system hosted; or both might be native.) C
- 725 ***device number**
- 726 The concept is handled in *stat()* §5.6.2 as ID of device.
- 727 **directory**
- 728 The format of the directory file is implementation defined, and differs radically
 729 between System V and 4.3BSD. However, routines (derived from 4.3BSD) for
 730 accessing directories are provided in Directory Operations §5.1.2 and certain
 731 constraints on the format of the information returned by those routines are made in
 732 Format of Directory Entries §5.1.1. 1)
- 733 **directory entry**
- 734 Throughout the document, the term *link* is used (about *link()* §5.3.4, for example)
 735 in describing the things that point to files from directories.
- 736 **dot** A
- 737 The symbolic name *dot* is carefully used in the standard to distinguish the working A
 738 directory filename from period or decimal point. A
- 739 **dot-dot**
- 740 Historical implementations permit the use of these filenames without their special
 741 meanings. Such use precludes any meaningful use of these filenames by a
 742 Conforming Application. Therefore such use is considered an extension, the use of
 743 which makes an implementation non-conforming. See also **pathname resolution**
 744 §B.2.4.
- 745 **Epoch** C
- 746 Normally, the origin of UNIX system time is referred to as "00:00:00 GMT, C
 747 January 1, 1970." Greenwich Mean Time is actually not a term acknowledged by C
 748 the international standards community therefore, this term, **Epoch**, is used to C
 749 abbreviate the reference to the actual standard, Coordinated Universal Time. The C
 750 concept of leap seconds is added for precision; at the time this standard was C
 751 published, 18 leap seconds had been added since January 1, 1970. These 18 C
 752 seconds are ignored to provide an easy and compatible method of computing time C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 753 differences. c
- 754 **FIFO special file**
- 755 See pipe §B.2.3.
- 756 **file**
- 757 It is permissible for an implementation defined file type to be non-readable or
758 non-writable.
- 759 **file classes** c
- 760 These classes correspond to the historical sets of permission bits. The classes are c
761 general to allow implementations flexibility in expanding the access mechanism c
762 for more stringent security environments. Note that a process is in one and only c
763 one class, so there is no ambiguity. c
- 764 **file system**
- 765 Historically the meaning of this term has been overloaded with two meanings: that
766 of the complete file hierarchy §B.2.4, and that of a mountable subset of that
767 hierarchy, i.e., a mounted file system §B.2.3. The standard uses the term **file**
768 **system** in the second sense, except that it is limited to the scope of a process (and a
769 process's root directory). This usage also clarifies the domain in which a file serial
770 number is unique.
- 771 ***group file**
- 772 Implementation defined; see Passwords §B.9.
- 773 ***historical implementations**
- 774 This term is used only in this appendix, not in the standard. It refers to
775 previously-existing implementations of programming interfaces and operating
776 systems that are related to the interface specified by the standard, especially to
777 those implementations described by the Base Documents §B.1.3. See also
778 Minimal Changes to Historical Implementations §B.1.2.8.
- 779 ***hosted implementation** c
- 780 This refers to a POSIX implementation that is accomplished through interfaces c
781 from the POSIX services to some alternate form of operating system kernel c
782 services. Note that the line between a hosted implementation and a native c
783 implementation is blurred, since most implementations will provide some services c
784 directly from the kernel, and others through some indirect path. (For example, c
785 *fopen()* might use *open()*; or *mkfifo()* might use *mknode()*.) There is no necessary c
786 relationship between the type of implementation and its correctness, performance, c
787 and/or reliability. c
- 788 ***implementation** c
- 789 The term is generally used instead of its synonym, **system**, to emphasize the c
790 consequences of decisions to be made by system implementors. Perhaps if no c
791 options or extensions to POSIX were allowed, this usage would not have occurred. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 792 ***incomplete path name**
 793 Absolute pathname §2.4 has been adequately defined.
- 794 ***kernel**
 795 See system call.
- 796 ***library routine**
 797 See system call.
- 798 ***logical device**
 799 Implementation defined.
- 800 ***mount point**
 801 The directory on which a **mounted file system** is mounted. This term, like
 802 *mount()* and *umount()* was not included because it was implementation defined.
- 803 ***mounted file system**
 804 See file system.
- 805 ***native implementation** c
 806 This refers to an implementation of POSIX that interfaces directly to an operating c
 807 system kernel addressed in the standard. See also **hosted implementation** §B.2.3 c
 808 and **cooperating implementation** §B.2.3. A similar concept from the UNIX world c
 809 is a native UNIX system, which would a be kernel derived from one of AT&T's c
 810 UNIX products. c
- 811 ***passwd file**
 812 Implementation defined; see **Passwords** §B.9.
- 813 **open file description** c
 814 An **open file description**, as it is currently named, “describes” how a file is being c
 815 accessed. What is currently called a **file descriptor** is actually just an identifier or c
 816 “handle;” it does not actually describe anything. c
- 817 The following alternate names were discussed: c
- 818 **open file description** c
 819 open instance, file access description, open file information, and file access c
 820 information. c
- 821 **file descriptor**
 822 file handle, file number [c.f., *fileno*].
- 823 **pipe**
 824 It proved convenient to define a **pipe** as a special case of a FIFO even though
 825 historically the latter were only introduced in System III and do not exist at all in c
 826 4.3BSD. c
- 827 **portable filename character set**
 828 The encoding of this character set is not specified: specifically, ASCII is not

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

829 required. But the implementation must provide a unique character code for each of
 830 the printable graphics specified by the standard. See also **filename portability**
 831 §B.2.4.

832 **regular file**

833 The standard does not intend to preclude the addition of structuring data (e.g.,
 834 record lengths) in the file, as long as such data is not visible to an application that
 835 uses the features described in the standard.

836 **root directory**

837 This definition permits the operation of *chroot()*, even though that function is not
 838 in the standard. See also **file hierarchy** §B.2.4.

839 ***root file system**

840 Implementation defined. A

841 ***root of a file system** A

842 Implementation defined. See **mount point**. A

843 **signal**

844 The definition implies a double meaning for the term. Although a signal is an
 845 event, common usage implies that a signal is an identifier of the event.

846 ***system call**

847 The distinction between a system call and a library routine is an implementation
 848 detail that may differ between implementations and has thus been excluded from
 849 the standard. See **Interface, Not Implementation** §B.1.2.2.

850 ***super-user** B

851 This concept, with great historical significance to UNIX system users, has been B
 852 replaced with the notion of **appropriate privileges**. B

853 **B.2.4 General Concepts**

854 **file access permissions**

855 A process should not try to anticipate the result of an attempt to access data by *a*
 856 *priori* use of these rules. Rather, it should make the attempt to access data and
 857 examine the return value (and possibly *errno*, as well), or use *access()* §5.6.3. An
 858 implementation may include other security mechanisms in addition to those
 859 specified in the standard, and an access attempt may fail because of those
 860 additional mechanisms even though it would succeed according to the rules given
 861 in this section. (For example, the user's security level might be lower than that of A
 862 the object of the access attempt.) The optional supplementary group IDs provide
 863 another reason for a process to not attempt to anticipate the result of an access
 864 attempt.

865 **file hierarchy**

866 Though the file hierarchy is commonly regarded to be a tree, the standard does not

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 867 define it as such for three reasons:
- 868 • As noted in the standard, links may join branches.
- 869 • In some network implementations, there may be no single absolute root
870 directory. See **pathname resolution**.
- 871 • With symbolic links (found in 4.3BSD), the file system need not be a
872 tree or even a Directed Acyclic Graph. C C
- 873 **file permissions** C
- 874 Examples of implementation defined constraints that may deny access are
875 mandatory labels and access control lists. C C
- 876 **filename portability**
- 877 Most historical implementations, including all of those described by the Base A
878 Documents §B.1.3, prohibit case folding in filenames, i.e., treating upper- and A
879 lowercase alphabetic characters as identical. However, some consider case folding A
880 desirable A
- 881 1. For user convenience. A
- 882 2. For ease of implementation of the standard interface as a hosted system on A
883 some popular operating systems, which is compatible with the goal of A
884 making the standard interface broadly implementable §B.1.2.7. A
- 885 Variants such as maintaining case distinctions in file names but ignoring them in A
886 comparisons have been suggested. Methods of allowing escaped characters of the A
887 case opposite the default have been proposed. A
- 888 Many reasons have been expressed for not allowing case folding, including: A
- 889 1. No solid evidence has been produced as to whether case sensitivity or case A
890 insensitivity is more convenient for users. A
- 891 2. Making case insensitivity a POSIX implementation option would be worse A
892 than either having it or not having it, because A
- 893 • More confusion would be caused among users. A
- 894 • Application developers would have to account for both cases in their A
895 code. A
- 896 • POSIX implementors would still have other problems with native file A
897 systems, such as short or otherwise constrained filenames, not to mention A
898 the lack of hierarchical directory structure. A
- 899 3. Case folding is not easily defined in many European languages, both because A
900 many of them use characters outside the USASCII alphabetic set, and A
901 because: A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 902 • In Spanish the digraph ll is considered to be a single letter, the A
 903 capitalized form of which may be either Ll or LL depending on context. A
- 904 • In French the capitalized form of a letter with an accent may or not retain A
 905 the accent depending on the country in which it is written. A
- 906 • In German the sharp ess may be represented as a single character A
 907 resembling a Greek beta (β) in lowercase but as the digraph SS in A
 908 uppercase. A
- 909 • In Greek there are several lowercase forms of some letters; the one to use A
 910 depends on its position in the word. Arabic has similar rules. A
- 911 4. Many East Asian languages, including Japanese, Chinese, and Korean, do A
 912 not distinguish case, and are sometimes encoded in character sets that use A
 913 more than one byte per character. A
- 914 5. Multiple character codes may be used on the same machine simultaneously. A
 915 There are several ISO character sets for European alphabets. In Japan, A
 916 several Japanese character codes are commonly used together, sometimes A
 917 even in filenames; this is evidently also the case in China. To handle case A
 918 insensitivity, the kernel would have to at least be able to distinguish for A
 919 which character sets the concept made sense. A
- 920 6. The file system implementation historically deals only with bytes, not with A
 921 characters, except for slash and the null byte. A
- 922 7. The purpose of the Working Group is to standardize the common, existing A
 923 definition §B.1.2.1 of the UNIX system programming interface, not to A
 924 change it. Mandating case insensitivity would make all historical A
 925 implementations non-standard. A
- 926 8. Not only the interface, but also application programs would need to change, A
 927 counter to the purpose of having minimal changes to existing application A
 928 code §B.1.2.9. A
- 929 9. At least one of the original developers of the UNIX system has expressed A
 930 objection in the strongest terms to either requiring case insensitivity or A
 931 making it an option, mostly on the basis that the standard should not hinder A
 932 portability of application programs across related implementations in order A
 933 to allow compatibility with unrelated operating systems. A

934 Two proposals were entertained regarding case folding in file names:

- 935 1. Remove all wording that previously permitted case folding.
 936 • Rationale: Case folding is inconsistent with portable filename character set
 937 definition and filename definition (all characters except slash and null). No
 938 known implementations allowing all characters except slash and null also do
 939 case folding.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

- 940 2. Change “though this practice is not recommended:” to “although this
941 practice is strongly discouraged”
942 **Rationale:** If case folding must be included in the standard, the wording
943 should be stronger to discourage the practice.
- 944 The consensus of the Working Group was in favor of proposal 1. Otherwise, a portable c
945 application would have to assume that case folding would occur when it wasn’t wanted, c
946 but that it wouldn’t occur when it was wanted. c
- 947 **file times update** c
948 **General Concepts** §2.4 has been changed to follow historical implementations. c
949 The times are not updated immediately, but are only marked for update by the c
950 functions. c
- 951 **pathname resolution**
952 What the filename **dot-dot** refers to relative to the root directory is
953 implementation defined. In Version 7 it refers to the root directory itself; this is
954 the behavior mentioned in the standard. In some networked systems the
955 construction `././hostname/` is used to refer to the root directory of another host,
956 and the standard permits this behavior.
- 957 Other networked systems use the construct `//hostname/` for the same purpose, i.e., A
958 a double initial slash is used. The Working Group decided to prohibit this practice, A
959 because if such a construction is not equivalent to a single leading slash, it is more A
960 difficult to write shell scripts that depend on concatenating a directory name with a A
961 filename part. The utility (and ubiquitousness) of such shell scripts was considered A
962 more important than a particular file system implementation. This consideration c
963 did not apply to `././hostname`, because that construct would not be used unless the c
964 application was deliberately accessing the network facility. c
- 965 The term **root directory** is only defined in the standard relative to the process. In
966 some implementations, there may be no absolute root directory. The initialization
967 of the root directory of a process is implementation defined.
- 968 **B.2.5 Error Numbers**
969 Checking the value of *errno* alone is not sufficient to determine the existence or type of
970 an error, since it is not required that a successful function call clear *errno*. The variable
971 *errno* should only be examined when the return value of a function indicates that the
972 value of *errno* is meaningful. In that case, the function is required to set the variable to
973 something other than zero.
- 974 A successful function call may set the value of *errno* to zero, or to any other value
975 (except where specifically prohibited: see `mkdir()` §B.5.4.1). But it is meaningless to do
976 so, since the value of *errno* is undefined except when the description of a function
977 explicitly states that it is set, and no function description states that it should be set on a
978 successful call. Most functions in most implementations do not change *errno* on

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

979 successful completion. Exceptions are *isatty()* §4.7.2 and *ptrace()*. The latter is not in
980 the standard, but is widely implemented and clears *errno* when called.

981 The standard requires (in the **Errors** subsections of function descriptions) certain error
982 values to be set in certain conditions because many existing applications depend on them.
983 Some error numbers, such as [EFAULT], are entirely implementation defined and are
984 noted as such in their description in **Error Numbers** §2.5. This section otherwise allows
985 wide latitude to the implementation in handling error reporting. All references to the
986 term **system call** have been excised from the descriptions of errors in this section.

987 Following each one-word symbolic name for an error, there is a one-line tag, which is
988 followed by a description of the error. The one-line tag is merely a mnemonic or
989 historical referent and is not part of the specification of the error. Many programs print
990 these tags on the standard error stream (often by using the Standard C *perror()* function)
991 when the corresponding errors are detected, but the standard does not require this action.

992 [EFAULT] Most historical implementations do not catch an error and set
993 *errno* when a bad address is given to the functions *wait()* §3.2.1,
994 *time()* §4.5.1, or *times()* §4.5.2. Some implementations cannot
995 reliably detect a bad address. And most systems that detect bad
996 addresses will do so only for a system call §B.2.3, not for a
997 **library routine** §B.2.3.

998 [EINTR] The standard does not prohibit implementations from restarting c
999 interrupted system calls, nor does it require that [EINTR] be c
1000 returned when another legitimate value may be substituted, e.g., a c
1001 partial transfer count when *read()* or *write()* are interrupted. c

1002 [ENAMETOOLONG] A

1003 [ENOMEM] The term **main memory** §B.2.3 has been eliminated from this
1004 description as being implementation defined.

1005 [ENOTTY] The symbolic name for this error is derived from a time when A
1006 a device control was done by *ioctl()* §B.2.5 and that operation was A
1007 only permitted on a terminal interface.

1008 B.2.6 Primitive System Data Types

1009 In early drafts, the standard specified that additional types that the implementation could
 1010 place into `<sys/types.h>` had to be named with a “_t” suffix. This restriction was
 1011 removed as it did not aid application portability and many implementations already were
 1012 in violation. c

1013 *clock_t* Traditionally, the type *time_t* was used for this. The Trial Use
 1014 Standard used *time_t*. The present type was adopted to match the
 1015 C Standard. See Time §B.4.5.

1016 *dev_t* This type may be made large enough to accomodate host-locality
 1017 considerations of networked systems.

1018 This type must be integral. Earlier drafts allowed this to be non- B
 1019 integral and provided a *samefile()* function for comparison. B

1020 *mode_t* This type was chosen so that implementations could choose the
 1021 appropriate integral type, and for compatibility with the
 1022 C Standard. 4.3BSD uses *unsigned short* and the *SVID* uses
 1023 *ushort*, which is the same thing. Historically, only the low-order c
 1024 sixteen bits are significant. c

1025 *nlink_t* This type was introduced in place of *short* for *st_nlink* §5.6.1 in
 1026 response to an objection that *short* was too small.

1027 *off_t* This type is used only in *lseek()* §6.5.3 and `<sys/stat.h>` §5.6.1. B
 1028 Many implementations would have difficulties if it were defined as
 1029 anything other than *long*. The Working Group realizes that
 1030 requiring an integral type limits the capabilities of *lseek()* to four
 1031 gigabytes. See *lread()* §B.6.4. Also, the C Standard supplies
 1032 routines that use larger types: see *fgetpos()* §B.6.5.3 and *fsetpos()* A
 1033 §B.6.5.3. A

1034 *pid_t* This type has been proposed, but was not approved by the A
 1035 Working Group, because *int* is in common use on known systems, A
 1036 and sufficient need for *pid_t* to justify cost of changes has not been A
 1037 demonstrated. Also, many applications assume the digital A
 1038 representation of a process ID has a maximum of five digits; thus a A
 1039 larger type would not be of much use without requiring change of A
 1040 all such applications. A

1041 *uid_t* Before the addition of this type, the data types used to represent
 1042 these values varied throughout the standard. The `<sys/stat.h>` B
 1043 §5.6.1 header defined these values as type *short*, the `<passwd.h>`
 1044 file (now `<pwd.h>` §9.2.2 and `<grp.h>` §9.2.1) used an *int* and
 1045 *getuid()* §4.2.1 returned an *int*. In response to a strong objection
 1046 to the inconsistent definitions, the Working Group decided to

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

1047 switch all the types to *uid_t*.

1048 In practice, those historical implementations that use varying types A
 1049 of this sort can typedef *uid_t* to short with no serious A
 1050 consequences. A

1051 The main problem associated with this change is a concern about
 1052 object compatibility after structure size changes. Since most
 1053 implementations will define *uid_t* as a short, the only substantive
 1054 change will be a reduction in the size of the passwd §9.2 structure.
 1055 Consequently, implementations with an overriding concern for
 1056 object compatibility can pad the structure back to its current size.
 1057 For that reason, this problem wasn't considered critical enough to
 1058 warrant the addition of a separate type to the standard.

1059 **B.2.7 Environment Description**

1060 **LC_*** LC_* acknowledges the fact that the interfaces presented in the c
 1061 draft are not complete and may be extended as new c
 1062 international functionality is required. In the ANSI X3J11 draft c
 1063 proposal, names preceded by "LC_" are reserved in the name c
 1064 space for future categories. c

1065 To avoid name clashes, new categories and environments c
 1066 variables will be divided into two classifications: c
 1067 implementation-independent and implementation-dependent. c

1068 Implementation-independent names will have the following c
 1069 format: c

1070 **LC_NAME** c

1071 where *NAME* is the name of the new category and environment c
 1072 variable. Capital letters must be used for implementation- c
 1073 independent names. c

1074 Implementation-dependent names must be in lower-case letter, c
 1075 as below: c

1076 **LC_name** c

1077 **PATH** Many historical implementations of the Bourne shell do not
 1078 interpret a trailing colon to represent the current working
 1079 directory, and are thus non-conforming. The C shell and the
 1080 Korn shell conform to the standard on this point. The usual
 1081 name of dot §2.3 may also be used to refer to the current
 1082 working directory.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 1083 **TZ** See *setlocale()* §8.1.2 for an explanation of the format. c
- 1084 **LOGNAME** 4.3BSD uses the environment variable **USER** for this purpose. c
- 1085 In most implementations, the value of such a variable is easily c
- 1086 forged, so security-critical applications should rely on other c
- 1087 means of determining user identity. c
- 1088 **B.2.8 C Language Definitions**
- 1089 The construct `<name.h>` for headers is also taken from the C Standard.
- 1090 **B.2.9 Numerical Limits**
- 1091 This section has been completely rewritten since the Trial Use Standard, in order to A
- 1092 clarify the scope and mutability of several classes of limits. A
- 1093 The standard does not require an application to include `<limits.h>` everywhere a limit in A
- 1094 it is used because many of them are system or application compile time constants that are A
- 1095 not useful at runtime. A
- 1096 If the translation and execution environments §B.1.4 are actually distinct, it may be A
- 1097 difficult to obtain information about runtime limits in the execution environment, A
- 1098 especially considering that the C Standard does not even require the limits of `<limits.h>` A
- 1099 to be kept in a file (they could instead be built into the translator). A useful technique is A
- 1100 to write a small application that does nothing when run but report back on relevant limits. A
- 1101 The language in the first paragraph about `#if` preprocessing directives is taken from the A
- 1102 C Standard. A
- 1103 **B.2.9.1 C Language Limits**
- 1104 See also C Language Definitions §2.8 and C Language, X3J11, and P1003.1 §B.1.4. A
- 1105 C
- 1106 {**CHAR_MIN**}
- 1107 It is possible to tell if the implementation supports native character A
- 1108 comparison as signed or unsigned by comparing this limit to zero. A
- 1109 {**WORD_BIT**}
- 1110 This limit has been omitted, as it is not referenced elsewhere in A
- 1111 POSIX. A
- 1112 No limits are given in `<limits.h>` for floating point values because none of the functions A
- 1113 in the standard proper use floating point values and all the functions that do that are A
- 1114 imported from the C Standard by **Referenced C Language Routines** §8.1 defined in the A
- 1115 C Standard, as are the limits that apply to the floating point values associated with them. A
- 1116 Though limits to the addresses to system calls were proposed, it is not clear how to A
- 1117 implement them for the range of systems being considered and, lacking a complete A
- 1118 proposal, the Working Group determined not to attempt this at this time. Limits A
- 1119 regarding hardware register characteristics were similarly proposed and not attempted. A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

1120 **B.2.9.2 Run-time Invariant Values**

1121 The criterion for inclusion of an item in this section is that a Conforming Application A
 1122 Using Extensions could break if the corresponding restriction is relaxed between the time A
 1123 the Conforming Application Using Extensions is translated and the time it is executed. A

1124 If, in a specific implementation, any of the parameters specified in this subsection can be A
 1125 varied at run time, the implementation will only be a conforming implementation when A
 1126 the values set at run time match those in the <limits.h> file. A

1127 The heading of the rightmost column of the table is given as “Minimum Value” rather A
 1128 than “Value” in order to emphasize that the numbers given in that column are minimal A
 1129 for the actual values a specific implementation is permitted to define in its <limits.h>. c
 1130 The values in the actual <limits.h> define, in turn, the maximum amount of a given c
 1131 resource that a Conforming Application can depend on finding when translated to A
 1132 execute on that implementation. A Conforming Application Using Extensions must A
 1133 function correctly even if the value given in <limits.h> is the minimum that is specified A
 1134 in the standard. (The application may still be written so that it performs more efficiently A
 1135 when a larger value is found in <limits.h>.) A conforming implementation must provide A
 1136 at least as much of a particular resource as that given by the value in the standard. An A
 1137 implementation that cannot meet this requirement (a “toy implementation”) cannot be a A
 1138 conforming implementation. A

1139 {FCHR_MAX} A

1140 is specifically a measure of the addressability of bytes in a file. It c
 1141 was dropped from the standard in Draft 12. The value given c
 1142 implies that *off_t* must be at least 24 bits wide. In terms of
 1143 testability, it should be possible to do the following on a
 1144 conforming implementation:

1145 Create a file with:

```
1146 int file;
1147 file = open(path, O_RDWR|O_CREAT|O_TRUNC, 0600);
1148 lseek(file, (off_t)16777215, SEEK_SET); B
1149 write(file, '1', 1); A
1150 lseek(file, (off_t)0, SEEK_SET); A
1151 /* read 16777215 bytes with value 0 */ B
1152 /* read 1 byte with value 1 */ A
```

1153 There is *no* requirement that a conforming implementation
 1154 provides the ability to create a non-sparse file containing 16777216 B
 1155 bytes (or any other number of bytes). It is expected, however, that A
 1156 it will be possible to configure specific instances of most specific A
 1157 implementations such that files of any required length less than or A
 1158 equal to {FCHR_MAX} + 1 can be created. A Conforming A
 1159 Application Using Extensions will generally depend on the ability A
 1160 to create non-sparse files of some specific length. It is the A

1161 responsibility of the administrator who configures a specific A
 1162 instance of a specific implementation to provide adequate file A
 1163 storage space to allow applications to run. To put this another A
 1164 way, even a Conforming Application Using Extensions will not A
 1165 run on a specific instance of a specific implementation if less file A
 1166 storage space is provided than is required by the Conforming A
 1167 Application Using Extensions. The standard says nothing about A
 1168 available file space, just as it says nothing about available memory A
 1169 space. A

1170 {MAX_INPUT} A
 1171 Since the only use of this limit is in relation to terminal input A
 1172 queues, it mentions them specifically. This limit was originally C
 1173 named {MAX_CHAR} in early drafts. Application writers should
 1174 use {MAX_INPUT} primarily as an indication of the number of
 1175 characters that can be written as a single unit by one Conforming
 1176 Application Using Extensions communicating with another via a
 1177 terminal device. It is *not* implied that input lines received from
 1178 terminal devices always contain {MAX_INPUT} characters or
 1179 fewer: an application that attempts to read more than
 1180 {MAX_INPUT} characters from a terminal may receive more than
 1181 {MAX_INPUT} characters.

1182 {PATH_MAX} A
 1183 A Conforming Application or Conforming Application Using
 1184 Extensions that, for example, compiles to use different algorithms
 1185 depending on the value of {PATH_MAX} should use code such as
 1186 #if defined(PATH_MAX) && PATH_MAX < 512 A
 1187 ... A
 1188 #else A
 1189 #if defined(PATH_MAX) /* PATH_MAX >= 512 */ A
 1190 ... A
 1191 #else /* PATH_MAX indeterminate */ A
 1192 ... A
 1193 #endif A
 1194 #endif A

1195 This is because the value tends to be very large or indeterminate
 1196 on most historical implementations (it is arbitrarily large on
 1197 System V). On such systems there is no way to quantify the limit,
 1198 and it seems counter-productive to include an artificially small
 1199 fixed value in <limits.h> in such cases.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

1200 B.2.9.3 Run-time Invariant Values (Possibly Indeterminate)

1201 B.2.9.4 Pathname Variable Values

1202 B.2.9.5 Run-time Increaseable Values

1203 Values appear in this section if there is no possibility that arbitrarily increasing them A
 1204 between the translation and the execution of a Conforming Application Using Extensions A
 1205 could break the Conforming Application Using Extensions. Specific instances of specific A
 1206 implementations may choose to increase the values in order to support non-portable A
 1207 applications. A

1208 Use of the word “may” in “...may increase the value” is correct. P1003.3 need not test A
 1209 whether the value is less restrictive than that given in <limits.h> or by how much. A

1210 A {DIR_LEVEL_MAX} limit was removed from the draft because it had no perceived C
 1211 value to an application. C

1212 B.2.9.6 Bounded Ranges of Values

1213 A Conforming Application can assume that it can have *at least* the most restrictive value A
 1214 of the resource. It has a “fighting chance” (a phrase used by P.J. Plauger of X3J11) of A
 1215 getting as much as that given by the least restrictive value. It can *never* get more than A
 1216 that given by the least restrictive value. The utility of the bounded range concept is that A
 1217 it allows the following: A

1218 a) If a Conforming Application wants (for example) to close all open files, the A
 1219 least restrictive value tells it how many close operations are needed in order A
 1220 to ensure that all files have been closed. Without knowledge of the value, A
 1221 this number is indeterminate. A

1222 b) The intention is that a supplier of a range of compatible computers should A
 1223 be able to ship a single <limits.h> which adequately describes the entire A
 1224 range. Thus if, for example, <limits.h> for a superminicomputer contains A
 1225 the pair A

```
1226 #define OPEN_MAX 20 A
1227 #define OPEN_MAX_CEIL 80 A
```

1228 an application running on the same vendor’s workstation is entitled to A
 1229 expect that it can have 20 open files (and may legitimately malfunction if it A
 1230 is not able to do so). The same binary application code, when running on a A
 1231 much larger member of the same machine family may find that it can have A
 1232 as many as 80 open files. An intelligently-written application may be able A
 1233 to optimize its algorithms according to the amount of a particular resource A
 1234 that it can obtain, but should not attempt to obtain more of any resource A
 1235 than that indicated by the corresponding upper limit defined by <limits.h>. A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 1236 Looking at the same issue from another angle, the vendor need only ship A
 1237 one C compiler package for the entire machine family; an application A
 1238 developer need only compile once to produce a program that runs optimally A
 1239 across the entire range of machines in the family. A
- 1240 Use of the word “may” in “may relax the corresponding restriction” is correct, but A
 1241 raises a testability issue. If, for example, `<limits.h>` suggests that it *may* be possible for a A
 1242 process to open as many as 80 files, but never to be able to open the eighty-first, P1003.3 A
 1243 must insist that this condition can be attained. A
- 1244 `{CHILD_MAX}` A
 1245 In a typical implementation, one process per user ID is used for the A
 1246 login shell, and one for the current process, leaving four potential A
 1247 children. A
- 1248 `{LOCK_MAX}` A
- 1249 `{PROC_MAX}` A
- 1250 `{SYS_OPEN}` These three limits were removed from `<limits.h>`. The B
 1251 information in `<limits.h>` should be useful to a Conforming B
 1252 Application; these three values were not useful: it is of no use, for A
 1253 example, for a Conforming Application to know the size of the A
 1254 system open file table, as there is no way that a process group, for A
 1255 instance, can ever be sure how many of those files it can open. A
 1256 The only thing that is certain is that each process in the group may A
 1257 be able to open no more than `{OPEN_MAX}` files, and may be able A
 1258 to open as many as `{OPEN_MAX_CEIL}`. `<limits.h>` implies this. A
 1259 `{SYS_OPEN}` does not add to the useful information available to A
 1260 the Conforming Application. A
- 1261 **B.2.10 Symbolic Constants**
- 1262 **B.2.10.1 Symbolic constants for the `access()` function**
- 1263 **B.2.10.2 Symbolic constants for the `lseek()` function**
- 1264 **B.2.10.3 Symbolic constants for portability specifications**
- 1265 **B.2.10.4 Compiler time symbolic constants for portability specifications**
- 1266 Related material appeared in an appendix of the Trial Use Standard. The purpose there A
 1267 was to allow an application developer to have a chance to determine whether a given A
 1268 application would run (or run well) on a given implementation. To this purpose has been A
 1269 added that of simplifying development of verification suites (see Verification Testing A
 1270 §A.2.3) for the standard. The constants given here were originally proposed for a A
 1271 separate file, `<posix.h>`, but the Working Group decided that they should appear in A
 1272 `<unistd.h>` along with other symbolic constants. A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

1273 **B.2.10.5 Execution time symbolic constants for portability specifications** A
 1274 Without the addition of `{_POSIX_NO_TRUNC}` and `_PC_NO_TRUNC` to the Configurable c
 1275 Open Variables list, the Standard says nothing about the effect of a pathname component c
 1276 longer than `{NAME_MAX}`. There are only two effects in common use in c
 1277 implementations: truncation, or an error. It is desirable to limit allowable behavior to c
 1278 these two cases. It is also desirable to permit applications to determine what an c
 1279 implementation's behavior is, because services that are available with one behavior may c
 1280 be impractical to provide with the other. However, since the behavior may vary from c
 1281 one file system to another, it may be necessary to use `pathconf()` to resolve it.

1282 B.3 Process Primitives

1283 B.3.1 Process Creation

1284 A common way to produce ("spawn") a descendant process that does not need to be c
 1285 waited on is to `fork()` to produce a child and `wait()` on the child. The child `fork()`s again c
 1286 to produce a grandchild. The child then exits and the parent's `wait()` returns. The c
 1287 grandchild is thus disinherited by its grandparent. c

1288 A simpler method (from the programmer's point of view) of spawning is to do c

```
1289     system("something &"); A
```

1290 However, this depends on features of a process (the shell) that are outside the scope of c
 1291 the present standard, although they may be addressed by P1003.2. c

1292 B.3.1.1 Process Creation

1293 During the `fork()` function call, signals directed to a group of processes, of which the c
 1294 child process is a member, may fail to be delivered to the child process. See `kill()` 9
 1295 §B.3.3.2. 9

1296 Many existing implementations have timing windows where a signal sent to a process A
 1297 group (e.g. an interactive SIGINT) just prior to or during execution of `fork()` is delivered A
 1298 to the parent following the `fork()` but not the child, because the `fork()` code clears the A
 1299 child's set of pending signals. It is not the intention of this standard to require, or even A
 1300 permit, this behavior. This behavior is only a consequence of the implementation failing A
 1301 to make the interval between signal generation and delivery totally invisible. From the A
 1302 application's perspective, a `fork()` call should appear atomic. A signal that is generated A
 1303 prior to the `fork()` should be delivered prior to the `fork()`. A signal sent to the process A
 1304 group after the `fork()` should be delivered to both parent and child. The implementation A
 1305 might actually initialize internal data structures corresponding to the child's set of A
 1306 pending signals to include signals sent to the process group during the `fork()`. Since the A
 1307 `fork()` call can be considered as atomic from the application's perspective, from that A
 1308 view the set would be initialized as empty and such signals would have arrived after the A
 1309 `fork()`. See also pending signals §B.3.3.6. A

1310 The [EINTR] error was considered too implementation-specific to include.

1311 B.3.1.2 Execute a File

1312 The value of *argc*, and the corresponding number of non-null *argv* pointers, should be 9
 1313 adjusted by the implementation so that *main()* receives at least one argument even when 9
 1314 the *exec()* call that invoked it supplied none. This is both because existing programs
 1315 expect it and also in order to conform with the C Standard.

1316 A Strictly Conforming Application §2.2.3 is required to supply an *arg0* that points to a
 1317 filename associated with the new process image file, and a Conforming Implementation
 1318 §2.2.1 is required to supply such an argument to *main()* in *argv[0]* (even if the calling
 1319 application did not). But no such requirement is placed on Application Conformance
 1320 §2.2.2, due to the use of the word “should” rather than “shall.”

1321 Some implementations provide a third argument to *main()* called *envp*. This is defined B
 1322 as a pointer to the environment. The C Standard provides *environ*, which replaces all B
 1323 need for the *envp* argument. Implementations are required to support the two-argument B
 1324 calling sequence, but this does not prohibit an implementation from supporting *envp* as B
 1325 an optional, third argument. B

1326 If the saved set-user-ID/saved set-group-ID option is implemented, *exec()* always saves B
 1327 the *uid* and *gid* of the process prior to the *exec()*. B

1328 [E2BIG] The limit {ARG_MAX} applies not just to the size of the argument
 1329 list, but to the sum of that and the size of the environment list.

1330 [EFAULT]

1331 Some existing systems return [EFAULT] rather than [ENOEXEC] c
 1332 when the new process image file is corrupted. They are non-
 1333 conforming.

1334 [ETXTBSY] The error [ETXTBSY] was considered too implementation- c
 1335 dependent to include. System V returns this error when the
 1336 executable file is currently open for writing by some process. The c
 1337 standard neither requires nor prohibits this behavior. c

1338 B.3.2 Process Termination

1339 “Abnormal termination with actions” includes, in most historical implementations, the c
 1340 creation of a file named *core* in the current working directory of the process. This file c
 1341 contains an image of the memory of the process, together with descriptive information c
 1342 about the process, perhaps sufficient to reconstruct the state of the process at the receipt c
 1343 of the signal. c

1344 There is a potential security problem in creating a *core* file if the process was set-user- c
 1345 ID and the current user is not the owner of the program, if the process was set-group-ID c
 1346 and none of the user’s groups match the group of the program, or if the user does not c
 1347 have permission to write in the current directory. In this situation, an implementation c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 1348 either should not create a `core` file or should make it unreadable by the user. c
- 1349 The name of the file is not mentioned in the standard because some historical c
 1350 implementations use a different name, such as by appending the process ID to the c
 1351 filename. However, applications are advised not to create files named `core` because of c
 1352 potential conflicts in many implementations. c
- 1353 **B.3.2.1 Wait for Process Termination**
- 1354 See `_exit()` §B.3.2.2. c
- 1355 The status values are given as specific bit encodings because they are that way in most
 1356 historical implementations and many existing programs expect it.
- 1357 A call on the `wait()` function only returns status on an immediate child process of the
 1358 calling process, i.e., a child that was produced by a single `fork()` §3.1.1 call (perhaps
 1359 followed by an `exec` §3.1.2 or other function calls) from the parent. If a child produces
 1360 grandchildren by further use of `fork()`, none of those grandchildren nor any of their
 1361 descendants will affect the behavior of a `wait()` from the original parent process.
- 1362 The `wait2()` function is provided for job control §B.3.3. It is identical to the `wait3()` A
 1363 function provided by 4.3BSD except that the third argument, the returned resource usage
 1364 summary, is not provided since it is not directly relevant to job control. The `wait2()`
 1365 function can be implemented as a library function on top of `wait3()`.
- 1366 Appendix E provides an alternative proposal for the `wait` family. Currently, there is no c
 1367 way to write a library routine, such as `system()` or `pclose()`, without interfering with c
 1368 other zombies. For example, consider the problem that which the P1003.2 group c
 1369 addressed: c
- ```

1370 stream = popen("/bin/true"); A
1371 (void) system("sleep 100"); A
1372 (void) pclose(stream); A

```
- 1373 On all systems since Version 6, the final `pclose()` will fail to reap the wait status of the c  
 1374 `popen()`. c
- 1375 This proposal changes section 3.2.1 by augmenting the `wait2()` call in several ways: c
- 1376 `wait2()` has been given a more descriptive name of `waitpid()`. c
- 1377 `waitpid()` can wait for a specific child, a child in the current process group, or a c  
 1378 child in a specific process group. The use of `pid` corresponds to the use of `pid` in c  
 1379 `kill()`. c
- 1380 `waitpid()` is required, and the WUNTRACED related actions are defined only for systems c  
 1381 that have the Job Control Option. c
- 1382 It should be noted that: c



- 1383                    `waitpid(stat_loc, -1, options)`                    A
- 1384 provides the same functionality as the function in the body of the standard:                    C
- 1385                    `wait2(stat_loc, options)`                    A
- 1386 The `waitpid()` function solves some major problems related to the functions `system()`, C  
 1387 `popen()`, and `pclose()` for Version 6, Version 7, Version 8, Version 9, System III, C  
 1388 System V, and 4BSD-based systems.                    C
- 1389 The `waitpid()` function would also greatly help in the writing of portable command C  
 1390 interpreters.                    C
- 1391 **B.3.2.2 Terminate a Process**
- 1392 The function `_exit()` is defined here instead of `exit()` because the C Standard defines the  
 1393 latter to have certain characteristics that are beyond the scope of the present standard,  
 1394 specifically the flushing of buffers on open files and the use of `atexit()`. See C Language  
 1395 and X3J11 §B.1.5. There are several public domain implementations of `atexit()` which  
 1396 may be of use to interface implementors who wish to incorporate it.
- 1397 It is important that the consequences of process termination as described in this section  
 1398 occur regardless of whether the process called `_exit()` (perhaps indirectly through `exit()`)  
 1399 or instead was terminated due to a signal or for some other reason. See also Process C  
 1400 Termination §B.3.2.                    C
- 1401 A language other than C may have other termination primitives than the C language  
 1402 `exit()` function, and programs written in such a language should use its native termination  
 1403 primitives, but those should have as part of their function the behavior of `_exit()` as  
 1404 described in this section. Implementations in languages other than C are outside the  
 1405 scope of the present standard, however.
- 1406 As required by X3J11, using `return()` from `main()` §3.1.2 is equivalent to calling `exit()` C  
 1407 with the same argument value. Also, reaching the end of the `main()` function is C  
 1408 equivalent to using `exit()` with an unspecified value.                    C
- 1409 Historically, the implementation-dependent process that inherits children whose parents C  
 1410 have terminated without waiting on them is called `init`, and has process ID 1.                    C
- 1411 The distinction between session process group leaders and job control process group  
 1412 leaders was created to allow the 4.2BSD semantics necessary to support job control  
 1413 without precluding the semantics of System V. System V sends the `SIGHUP` signal to the  
 1414 process group of a terminating process group leader. Such a process group leader is  
 1415 typically a login shell. 4.2BSD does not send `SIGHUP` under these conditions for two  
 1416 reasons:
- 1417 • First, job control semantics preclude killing background jobs at logout. While  
 1418 System V provides the `nohup` command to prevent killing background processes at  
 1419 logout, the user must make the decision when launching the command. The point of  
 1420 job control is that such decisions can be changed after launching the command.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

1421 • Second, every command pipeline launched by a job control shell (such as `cs`)  
 1422 resides in its own unique process group with one command in the pipeline being the  
 1423 process group leader. If `SIGHUP` were sent to the process group when that process  
 1424 terminated, the remaining pipeline would be prematurely terminated.

1425 If the terminating process has any children which are currently stopped, those children  
 1426 will be sent `SIGHUP` immediately followed by `SIGCONT`. This continues the stopped  
 1427 children and, unless they are catching or ignoring `SIGHUP`, also causes them to terminate.  
 1428 The goal is to prevent stopped processes from languishing forever. When a process exits  
 1429 with stopped children, those children are no longer under the control of a job control  
 1430 shell and hence would not normally ever be continued. See also the discussion of  
 1431 sending `SIGKILL` to stopped orphaned processes in Signal Names §B.3.3.1.

### 1432 B.3.3 Signals

1433

1434 Signals, as defined in the Trial Use Standard, and in Version 7, System III, the 1984  
 1435 */usr/group Standard*, and System V (except very recent releases), have shortcomings  
 1436 which make them unreliable for many application uses. Several objections have been  
 1437 voiced to the Trial Use Standard because of this. Therefore a new signal mechanism,  
 1438 based very closely on the one of 4.2BSD and 4.3BSD, was added to the standard. With  
 1439 the exception of two features (see item 4 below and also Examine Pending Signals  
 1440 §B.3.3.6), it is possible to implement the POSIX interface as a simple library veneer on  
 1441 top of 4.3BSD.

1442 The major differences from the BSD mechanism are:

#### 1443 1. Signal mask type.

1444 BSD uses the type `int` to represent a signal mask, thus limiting the number of  
 1445 signals to the number of bits in an `int` (typically thirty-two). The new standard  
 1446 instead uses a defined type for signal masks. Because of this change, the interface  
 1447 is significantly different than in BSD implementations, although the functionality  
 1448 and potentially the implementation are very similar.

#### 1449 2. Restarting system calls.

1450 Unlike all previous historical implementations, 4.2BSD restarts some interrupted  
 1451 system calls rather than returning an error with `errno` set to `[EINTR]` after the  
 1452 signal-catching function returns. This change caused problems for some existing  
 1453 application code. 4.3BSD and other systems derived from 4.2BSD allow the  
 1454 application to choose whether system calls are to be restarted. The standard (in  
 1455 *sigaction()* §3.3.4) does not require restart of functions, because it was not clear  
 1456 that the semantics of system call restart in any existing implementation were useful  
 1457 enough to be of value in a standard. Implementors are free to add such  
 1458 mechanisms as extensions.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.



- 1459    3. **Signal stacks.**
- 1460       The 4.2BSD mechanism includes a function *sigstack()*. The 4.3BSD mechanism
- 1461       includes this and a function *sigreturn()*. No equivalent is included in the standard
- 1462       because these functions are not clearly portable or necessary. See also **Non-local**
- 1463       **Jumps §8.4.**
- 1464    4. **Pending signals.**
- 1465       The *sigpending()* §3.3.6 function is the sole new signal operation introduced in the c
- 1466       standard. It was requested by some members of the Working Group and was seen
- 1467       as a simple and useful feature.
- 1468    The Working Group considered making reliable signals optional. However, the A
- 1469    consensus was that this would hurt application portability, as a large percentage of A
- 1470    applications using signals can be hurt by the unreliable aspects of the *signal()* §B.8.3.2 A
- 1471    mechanism. This unreliability stems from the specification that the signal action is reset c
- 1472    to SIG\_DFL before the user's signal-catching routine is entered. c
- 1473    Most traditional implementations do not queue signals, i.e., a process's signal handler is c
- 1474    invoked once, even if the signal has been generated multiple times before it is delivered. c
- 1475    A notable exception to this is SIGCLD which, in System V, is queued. The Working c
- 1476    Group decided to neither require nor prohibit the queueing of signals. It is expected that c
- 1477    a future Real Time Extension to this standard (see **Real Time Extensions §A.2.4**) will c
- 1478    address the issue of reliable queueing of event notification. c
- 1479    Note that an application which simply catches the interactive SIGINT signal with *signal()* A
- 1480    can be terminated with no chance to recover when two such signals arrive sufficiently A
- 1481    close in time (e.g., when a user gets impatient on a busy system). A
- 1482    **Job Control.** A
- 1483    The intent in adding 4.2BSD-style job control functionality was to adopt the necessary A
- 1484    4.2BSD programmatic interface with only minimal changes to resolve syntactic or A
- 1485    semantic conflicts with System V or to close recognized security holes. The goal was to A
- 1486    maximize the ease of providing both conforming implementations and Conforming A
- 1487    Applications. A
- 1488    Discussions of the changes can be found in the sections which discuss the specific A
- 1489    interfaces. See sections: **Wait for Process Termination §B.3.2.1, Terminate a A**
- 1490    **Process §B.3.2.2, Signal Names §B.3.3.1, Send a Signal to a Process §B.3.3.2, A**
- 1491    **Examine and Change Signal Action §B.3.3.4, Set Process Group §B.4.3.2, Job A**
- 1492    **Access Control §B.7.1.1.5, and Set Distinguished Process Group ID §B.7.2.4. A**
- 1493    It is only useful for a process to be affected by job control signals if it is the descendant A
- 1494    of a job control shell. Otherwise, there will be nothing which continues the stopped A
- 1495    process. Because a job control shell is allowed, but not required, by the standard, an A
- 1496    implementation must provide a mechanism which shields processes from job control A
- 1497    signals when there is no job control shell. The usual method is for the system A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.



1498 initialization process (typically called `init`), which is the ancestor of all processes, to A  
1499 launch its children with the signal handling action set to `SIG_IGN` for the signals A  
1500 `SIGTSTP`, `SIGTTIN`, and `SIGTTOU`. Thus all login shells start with these signals ignored. A  
1501 If the shell is not job control cognizant, then it should not alter this setting and all its A  
1502 descendants should inherit the same ignored settings. At the point where a job control A  
1503 shell is launched, it resets the signal handling action for these signals to be `SIG_DFL` for A  
1504 its children and (by inheritance) their descendants. Also, shells which are not job control A  
1505 cognizant will not alter the process group of their descendants or of their controlling A  
1506 terminal; this has the effect of making all processes be in the foreground (assuming the A  
1507 shell is in the foreground). A

1508 POSIX does not specify how controlling terminal access is affected by a user logging out A  
1509 (that is, by a login shell terminating). 4.2BSD uses the `vhangup()` function to prevent any A  
1510 access to the controlling terminal through file descriptors opened prior to logout. A  
1511 System V does nothing to prevent controlling terminal access through file descriptors A  
1512 opened prior to logout (except for the case of the special file, `/dev/tty`). Some A  
1513 implementations choose to make processes immune from job control after logout (that is, A  
1514 such processes are always treated as if in the foreground); other implementations A  
1515 continue to enforce foreground/background checks after logout. Therefore, a A  
1516 Conforming Application should not attempt to access the controlling terminal after A  
1517 logout since such access is unreliable. A

### 1518 B.3.3.1 Signal Names

#### 1519 B.3.3.1.1 Synopsis C

#### 1520 B.3.3.1.2 Description C

1521 The restriction on the actual type used for `sigset_t` is intended to guarantee that these A  
1522 objects can always be assigned, have their address taken, and be passed as parameters by A  
1523 value. It is not intended that this type be a structure including pointers to other data A  
1524 structures, as that could impact the portability of applications performing such A  
1525 operations. A reasonable implementation could be a structure containing an array of A  
1526 some integer type. A

1527 The signals described in the document must have unique values so that they may be c  
1528 named as parameters of `case` statements in the body of a C language `switch` clause. c  
1529 However, implementation defined signals may have values that overlap with each other c  
1530 or with signals specified in this document. An example of this is `SIGABRT`, which c  
1531 traditionally overlaps some other signal, such as `SIGIOT`. c

1532 `SIGKILL`, `SIGTRAP`, `SIGUSR1`, and `SIGUSR2` are ordinarily generated only through the c  
1533 explicit use of the `kill()` function, although some implementations generate `SIGKILL` c  
1534 under extraordinary circumstances. `SIGTERM` is traditionally the default signal sent by A  
1535 the `kill` command. A

1536 The signals `SIGBUS`, `SIGEMT`, `SIGIOT`, `SIGTRAP`, and `SIGSYS` were omitted from the c  
1537 standard because their behavior is implementation dependent and could not be c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

1538 adequately categorized. Conforming implementations may deliver these signals, but c  
1539 must document the circumstances under which they are delivered and note any c  
1540 restrictions concerning their delivery. c

1541 The signals SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, and SIGCONT are provided for job c  
1542 control and are unchanged from 4.2BSD. The signal SIGCLD is also typically used by c  
1543 job control shells to detect children which have terminated or, as in 4.2BSD, stopped. c  
1544 However, the 4.2BSD name, SIGCHLD, was dropped in favor of the System V SIGCLD. c  
1545 See also SA\_CLDSTOP §B.3.3.4. B

1546 The signals SIGUSR1 and SIGUSR2 are commonly used by applications for notification of A  
1547 exceptional behavior and are described as “reserved as application defined” so that such A  
1548 use is not prohibited. Implementations should not generate SIGUSR1 or SIGUSR2, except c  
1549 when explicitly requested by *kill()* §3.3.2. It is recommended that libraries not use these A  
1550 two signals, as such use in libraries could interfere with their use by applications calling A  
1551 the libraries. If such use is unavoidable it should be documented. It is prudent for non- A  
1552 portable libraries to use non-standard signals to avoid conflicts with use of standard A  
1553 signals by portable libraries. A

1554 In actual existing implementations, there are a few cases where the interval between c  
1555 generation and delivery of unmasked signals is visible to applications. For example, a c  
1556 pending signal (masked or unmasked) is discarded when its signal action is set to c  
1557 SIG\_IGN. Implementations should make this interval invisible to the extent possible. c  
1558 When this is totally true, references to pending signals apply only to pending, masked c  
1559 signals. c

1560 There is one case where a blocked signal does not remain pending until it is unblocked. c  
1561 In the System V implementation of *signal()*, there are some cases in which pending c  
1562 signals are also discarded when the action is set to SIG\_DFL or a signal-catching routine. c

1563 In 4.2BSD and 4.3BSD, there is one other case where a blocked signal is not kept c  
1564 pending. When the signal is being ignored and is also blocked, it is discarded c  
1565 immediately on generation. The Working Group did not wish to standardize this c  
1566 behavior. Implementations which do this do not conform completely to this standard. c

1567 There is very little if anything that a Conforming Application can do by catching, A  
1568 ignoring, or masking any of the signals SIGILL, SIGTRAP, SIGIOT, SIGEMT, SIGBUS, A  
1569 SIGSEGV, SIGSYS, or SIGFPE. They will generally be generated by the system only in B  
1570 cases of programming errors. While it may be desirable for some robust code (e.g., a B  
1571 library routine) to be able to detect and recover from programming errors in other code, B  
1572 these signals are not nearly sufficient for that purpose. One portable use that does exist B  
1573 for these signals is that a command interpreter can recognize them as the cause of a B  
1574 process’s termination (with *wait()*) and print an appropriate message. The mnemonic B  
1575 tags for these signals are derived from their PDP-11 origin. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.



1576 **B.3.3.1.3 Signal Actions**

1577 There is no portable way for an application to catch or ignore non-standard signals. A  
 1578 Some implementations define the range of signal numbers, so applications can install A  
 1579 signal catching functions for all of them. Unfortunately implementation defined signals A  
 1580 often cause problems when caught or ignored by applications that do not understand the A  
 1581 reason for the signal. While the desire exists for an application to be more robust by A  
 1582 handling all possible signals (even those only generated by *kill()*), no existing A  
 1583 mechanism was found to be sufficiently portable to include in the standard. The value of A  
 1584 such a mechanism, if included, would be diminished given that SIGKILL would still not A  
 1585 be catchable. A

1586 C

1587 The specification of the effects of SIG\_IGN on SIGCLD as implementation defined c  
 1588 permits but does not require the System V effect of causing terminating children to be c  
 1589 ignored by *wait()* §3.2.1. Yet it permits SIGCLD to be effectively ignored in an c  
 1590 implementation-independent manner by use of SIG\_DFL. c

1591 Some implementations (System V, for example) assign different semantics for SIGCLD c  
 1592 depending on whether the action is set to SIG\_IGN or SIG\_DFL. Since the standard c  
 1593 requires that the default action for SIGCLD be to ignore the signal, applications should c  
 1594 always set the action to SIG\_DFL in order to avoid SIGCLD. c

1595 Some implementations (System V, for example) will deliver a SIGCLD signal B  
 1596 immediately when a process establishes a signal-catching function for SIGCLD when that B  
 1597 process has a child that has already terminated. Other implementations, such as 4.3BSD, B  
 1598 do not generate a new SIGCLD signal in this way. In general, a process should not c  
 1599 attempt to alter the signal action for the SIGCLD signal while it has any outstanding c  
 1600 children. c

1601 SIGCONT has no effect on a running process if the action is set to SIG\_DFL, even though  
 1602 the signal will still cause a stopped process to continue.

1603 If a process is orphaned (because its parent has terminated) and then subsequently stops,  
 1604 it is no longer under the control of a job control shell and hence would not normally ever  
 1605 be continued. Because of this, orphaned processes which stop are sent the SIGKILL signal  
 1606 which causes them to terminate. The goal is to prevent stopped processes from  
 1607 languishing forever. See also SIGCONT §B.3.3.1. A

1608 In order to prevent errors arising from interrupting non-reentrant function calls, B  
 1609 applications should protect calls to these functions either by blocking the appropriate B  
 1610 signals or through the use of some programmatic semaphore. The standard does not c  
 1611 address the more general problem of synchronizing access to shared data structures. c  
 1612 Naturally, the same principles apply to the reentrancy of application routines and c  
 1613 asynchronous data access. Note that *longjmp()* is not in the list of reentrant functions; B  
 1614 applications that *longjmp()* out of signal handlers require rigorous protection in order to B  
 1615 be portable. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.



### 1616 B.3.3.2 Send a Signal to a Process

1617 The semantics for permission checking for *kill()* differ between System V and most other A  
 1618 implementations, such as Version 7 or 4.3BSD. The semantics chosen for the standard A  
 1619 agree with System V. Specifically, a setuid process cannot protect itself against signals A  
 1620 (or at least not against SIGKILL) unless it changes its real user ID. This choice allows the A  
 1621 user who starts an application to send it signals even if it changes its effective user ID. A  
 1622 The other semantics give more power to an application that wants to protect itself from A  
 1623 the user who ran it. A

1624 The implementation defined processes to which a signal cannot be sent may include the  
 1625 scheduler or *init*.

1626 As in 4.2BSD, the SIGCONT signal can be sent to any descendant process regardless of  
 1627 user ID security checks. This allows a job control shell to continue a job even if  
 1628 processes in the job have altered their user IDs (as in the *su* command). Note that this  
 1629 applies to all descendant processes, not just immediate children. A similar relaxation of  
 1630 security is not necessary for the other job control signals since those signals are typically  
 1631 sent by the terminal driver in recognition of special characters being typed; the terminal  
 1632 driver bypasses all security checks.

1633 In secure implementations, a process may be restricted from sending a signal to a process c  
 1634 having a different security label. In order to prevent the existence or non-existence of a c  
 1635 process from being used as a covert channel, such processes should appear non-existent c  
 1636 to the sender; i.e., [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to c  
 1637 such processes. c

### 1638 B.3.3.3 Manipulate Signal Sets

1639 The implementation of the *siginitset()* function may reasonably be a no-op. It is also A  
 1640 reasonable for it to initialize part of the structure, such as a version field, to permit binary A  
 1641 compatibility between releases where the size of the set varies. This function is not A  
 1642 intended for dynamic allocation. A

### 1643 B.3.3.4 Examine and Change Signal Action

1644 There was a proposal to change the declared type of the signal handler to: c

```
1645 void func (int sig, ...); A
```

1646 The ellipsis (“, ...”) is Standard C syntax to indicate a variable number of arguments. c  
 1647 Its use was intended to allow the implementation to pass additional information to the c  
 1648 signal handler in a standard manner. c

1649 Unfortunately, this construct would require all signal handlers to be defined with this c  
 1650 syntax, because the C Standard allows implementations to use a different parameter c  
 1651 passing mechanism for variable parameter lists than for non-variable parameter lists. c  
 1652 Thus all existing signal handlers in all existing applications would have to be changed to c  
 1653 use the variable syntax in order to be standard and to be portable. This is in conflict with c  
 1654 the goal of minimal changes to existing application code §B.1.2.9. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 1655 This problem with variable parameter lists does not apply to *open()*, *execl()*, *printf()*, c  
 1656 and other functions written by implementor of Standard C or POSIX. The application c  
 1657 developer does not have to provide a function parameter type definition of these c  
 1658 functions, and the declaration used by the implementor of the standard will determine the c  
 1659 mechanism used for passing variable argument lists. c
- 1660 The problem would also not occur for new facilities, since application writers could use c  
 1661 the appropriate function parameter definition in their new code. c
- 1662 The Working Group has nonetheless chosen to avoid the use of variable argument syntax c  
 1663 and of function parameter types in general in order to ease bindings of POSIX to c  
 1664 languages other than Standard C. See Conformance §B.2.2 and Function parameter c  
 1665 type lists §B.1.4. c
- 1666 The SA\_CLDSTOP flag, when supplied in the *sa\_flags* parameter, allows overloading c  
 1667 SIGCLD with the 4.2BSD SIGCHLD semantics necessary for job control. c
- 1668 **B.3.3.5 Examine and Change Blocked Signals** c  
 1669 c
- 1670 **B.3.3.6 Examine Pending Signals** c  
 1671 c
- 1672 **B.3.3.7 Wait for a Signal** c  
 1673 Normally, at the beginning of a critical code section, a specified set of signals is blocked B  
 1674 using the *sigprocmask()* function. When the process has completed the critical section B  
 1675 and needs to wait for the previously blocked signal(s), it pauses by calling *sigsuspend()* B  
 1676 with the mask that was returned by the *sigprocmask()* call. B
- 1677 **B.3.4 Timer Operations**
- 1678 **B.3.4.1 Process Alarm Clock**
- 1679 Because many traditional implementations (including Version 7 and System V) do allow A  
 1680 an alarm to occur up to a second early, the Working Group did not feel it could disallow A  
 1681 this behavior, and thus a Conforming Application needs to be prepared for it. However, A  
 1682 the Working Group does not want to encourage this behavior. Other implementations A  
 1683 allow alarms up to half a second early, up to 1/{CLK\_TCK} seconds early, or do not A  
 1684 allow them to occur early at all. The latter is considered most appropriate. Future real- c  
 1685 time standards related to this one (see Real Time Extensions §A.2.4) may specify such c  
 1686 facilities. c



1687 **B.3.4.2 Suspend Process Execution**

1688 Many common uses of *pause()* have timing windows. The scenario involves checking a A  
 1689 condition related to a signal and, if the signal has not occurred, calling *pause()*. When A  
 1690 the signal occurs between the check and the call to *pause()*, the process often blocks A  
 1691 indefinitely. The *sigprocmask()* and *sigsuspend()* functions can be used to avoid this A  
 1692 type of problem. A

1693 **B.3.4.3 Delay Process Execution**

1694 Traditional implementations often implement *sleep()* using *alarm()* and *pause()*. One A  
 1695 such implementation is prone to infinite hangs as described in *pause()* §B.3.4.2. Another A  
 1696 such implementation uses the C language *setjmp()* and *longjmp()* functions to avoid that A  
 1697 window. That implementation introduces a different problem; when the alarm signal A  
 1698 interrupts a signal catching function installed by the user to catch a different signal the A  
 1699 *longjmp()* aborts that signal-catching function. An implementation based on A  
 1700 *sigprocmask()*, *alarm()*, and *sigsuspend()* can avoid these problems. A

1701 Scheduling delays may cause the process to return from the *sleep()* function significantly B  
 1702 after the requested time. In such cases, the return value should be set to zero, since the B  
 1703 formula (requested time minus the time actually spent) yields a negative number and B  
 1704 *sleep()* returns an **unsigned int**. B

1705 **B.4 Process Environment**1706 **B.4.1 Process Identification**1707 **B.4.1.1 Get Process and Parent Process IDs**1708 **B.4.2 User Identification**1709 **B.4.2.1 Get Real User, Effective User, Real Group, and Effective Group IDs**1710 **B.4.2.2 Set User and Group IDs**

1711 Another way of looking at the behavior of these two functions: 9

1712 The call *setuid(uid)* shall result in both the real user ID and the effective user ID 9  
 1713 being equal to *uid* if: 9

1714 the effective user ID is super-user 9

1715 *or* 9

1716 the real user ID is *uid* 9

1717 *or* 9

1718 the effective user ID is *uid* (implementation permitting). 9

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.



- 1719 The call *setgid(gid)* shall result in both the real group ID and the effective user ID 9  
 1720 being equal to *gid* if: 9
- 1721 the effective user ID is super-user 9
  - 1722 *or* 9
  - 1723 the real group ID is *gid* 9
  - 1724 *or* 9
  - 1725 the effective group ID is *gid* (implementation permitting). 9
- 1726 The call *setuid(uid)* sets the effective user ID of the calling process to *uid* if any of the B  
 1727 following conditions are met: B
- 1728 The real user ID of the calling process is *uid*. B
  - 1729 The implementation provides the saved set-user-ID option and the saved set- B  
 1730 user-ID for the calling process is *uid*. B
  - 1731 The process has appropriate privileges. In this case, the real user ID and optional B  
 1732 saved set-user-ID are also set to *uid*. B
- 1733 The saved set-user-ID capability allows a program to regain the effective user ID B  
 1734 established at the last *exec* §3.1.2 call. Similarly, the saved set-group-ID capability B  
 1735 allows a program to regain the effective group ID established at the last *exec* call. B
- 1736 These last two capabilities are derived from System V. Without them, a program may  
 1737 have to run as super-user in order to perform the same functions, because super-user can  
 1738 write on the user's files. This is a problem because such a program can write on *any*  
 1739 user's files, and so must be carefully written to emulate the permissions of the calling  
 1740 process properly.
- 1741 The ability to set the real user ID to the value of its effective user ID corresponds to the B  
 1742 behavior of 4.2BSD and 4.3BSD. This is not a security risk over systems that do not B  
 1743 implement it; it actually reduces the access capabilities of a process. B
- 1744 **B.4.2.3 Get Supplementary Group IDs**
- 1745 The related function *setgroups()* §B.4.2.3 is a privileged operation and therefore is not 9  
 1746 covered by this standard. 9
- 1747 The wording regarding the group of a newly created regular file, directory, or FIFO in  
 1748 *open()* §5.3.1, *mkdir()* §5.4.1, *mkfifo()* §5.4.2, respectively, uses “may” rather than  
 1749 “shall” in order to permit both the System V (and Version 7) behavior (in which the  
 1750 group of the new object is set to the effective group ID of the creating process) and the  
 1751 4.3BSD behavior (in which the new object has the group of its parent directory). An A  
 1752 application that needs a file to be created in the group of the effective group ID should A  
 1753 use *chown()* §5.6.5 to ensure the new group regardless of the style of groups the interface A  
 1754 implements. A

1755 **B.4.2.4 Get User Name**

1756 **L\_cuserid** must be defined appropriately for a given implementation and must be greater c  
 1757 than zero so that array declarations using it are accepted by the compiler. The value c  
 1758 includes the terminating null byte. c

1759 **B.4.3 Process Groups**1760 **B.4.3.1 Get Process Group ID**

1761 4.3BSD provides a *getpgrp()* function that returns the process group ID for a specified c  
 1762 process. Although this function is used to support job control, all known job control c  
 1763 shells always specify the calling process with this function. Thus the simpler System.V c  
 1764 *getpgrp()* suffices and the added complexity of the 4.3BSD *getpgrp()* has been omitted c  
 1765 from the standard. c

1766 **B.4.3.2 Set Process Group ID**

1767 .br c

1768 **B.4.3.3 Set Process Group ID for Job Control**

1769 The *jcsetpgrp()* function is similar to the *setpgrp()* function of 4.2BSD. The differences B  
 1770 are:

1771 4.2BSD *setpgrp()* allows the caller to specify the process ID of the process to affect.  
 1772 Since all known job control shells always affect the calling process, this parameter was  
 1773 deleted; the affected process is now always the calling process.

1774 4.2BSD allowed the specified new process group to assume any value. This presents c  
 1775 certain security problems and is more flexible than necessary to support job control. In c  
 1776 keeping with the new security model (see Job Control §B.3.3), *jcsetpgrp()* only allows c  
 1777 the calling process to join a process group that is already associated with the calling c  
 1778 process' controlling terminal. One special case is where the calling process is creating a c  
 1779 new process group, that is where there are no other processes currently in the process c  
 1780 group. In this case, the calling process is allowed to join the new group. c

1781 These restrictions maintain the assertion that the calling process is not introducing a new c  
 1782 (different) controlling terminal into an already existing process group. Violating this c  
 1783 assertion would result in one process group (or job) which could be controlled by more c  
 1784 than one controlling terminal (or login session). The typical scenario that is being c  
 1785 prevented is for a process to first use *jcsetpgrp()* to join the process group of another c  
 1786 login session and then to use *tcsetpgrp()* §7.2.4 to allow keyboard signals from its c  
 1787 controlling terminal to affect processes in a different session. c

1788 One non-obvious use of *jcsetpgrp()* is to allow a job control shell to return itself to its c  
 1789 original process group (the one in effect when the job control shell was executed). A job c  
 1790 control shell does this before returning control back to its parent when it is terminating or c  
 1791 suspending itself as a way of restoring its job control "state" back to what its parent c  
 1792 would expect. (Note that the original process group of the job control shell typically c  
 1793 matches the process group of its parent, but this is not necessarily always the case.) See c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.



1794 also *tcsetpgrp()* §B.7.1.7. C

## 1795 B.4.4 System Identification

### 1796 B.4.4.1 System Name

1797 The values of the structure members are not constrained to have any relation to the  
 1798 version of this interface standard implemented in the operating system. An application  
 1799 implementor should instead depend on `{_POSIX_VERSION}` and related constants C  
 1800 defined in Symbolic Constants §2.10. C

1801 The standard does not define the sizes of the members of the structure and permits them  
 1802 to be of different sizes, although most implementations define them all to be the same  
 1803 size: eight bytes plus one byte for the string terminator. That size for *nodename* is not  
 1804 enough for use with many networks.

1805 The *uname()* function is specific to System III, System V, and related implementations,  
 1806 and it does not exist in Version 7 or 4.3BSD. The values it returns are set at system A  
 1807 compile time in those existing implementations.

1808 4.3BSD has *gethostname()* and *gethostid()*, which return a symbolic name and a numeric  
 1809 value, respectively. There are related *sethostname()* and *sethostid()* functions that are  
 1810 used to set the values the other two functions return. The length of the host name is  
 1811 limited to 31 characters in most implementations and the host ID is a thirty-two bit  
 1812 integer.

### 1813 B.4.5 Time

1814 The *time()* §4.5.1 function returns a value in seconds (type *time\_t*) while *times()* §4.5.2  
 1815 returns a set of values in `{CLK_TCK}`ths of a second (type *clock\_t*).

1816 Some historical implementations, such as 4.3BSD, have mechanisms capable of returning A  
 1817 more precise times (see *gettimeofday()* §B.4.5.1). A generalized timing scheme to unify A  
 1818 these various timing mechanisms has been proposed but not adopted in this standard; see C  
 1819 Real Time Extensions §A.2.4. C

#### 1820 B.4.5.1 Get System Time

1821 Implementations in which *time\_t* is a thirty two bit signed integer (most historical  
 1822 implementations) will fail in the year 2038. The Working Group chose not to try to fix  
 1823 this. But they did require the use of *time\_t* in order to ease the eventual fix. A

1824 Many historical implementations (including Version 7) and the 1984 *usr/group Standard*  
 1825 use long instead of *time\_t*. The present standard uses the latter type in order to agree  
 1826 with the C Standard.

1827 4.3BSD includes *time()* only as an interface to the more flexible *gettimeofday()* §B.4.5.1  
 1828 function.



1829 **B.4.5.2 Process Times**

1830 The inclusion of times of child processes is recursive, so that a parent process may  
 1831 collect the total times of all of its descendants. But the times of a child are only added to  
 1832 those of its parent when its parent successfully waits on the child. Thus it is not  
 1833 guaranteed that a parent process will always be able to see the total times of all its  
 1834 descendants.

1835 c

1836 If the type *clock\_t* is defined to be a signed thirty-two bit integer, it will overflow in  
 1837 somewhat more than a year if {CLK\_TCK} is 60, or less than a year if it is 100. There  
 1838 are individual systems that run continuously for longer than that. The standard permits  
 1839 an implementation to make the reference point for the returned value be the startup time  
 1840 of the process, rather than system startup time.

1841 **B.4.6 Environment Variables**1842 **B.4.6.1 Environment Access**

1843 Additional functions *putenv()* and *clearenv()* were considered but rejected because they A  
 1844 were more oriented towards system administration than ordinary application programs. A

1845 **B.4.7 Terminal Identification**

1846 The difference between *ctermid()* and *ttyname()* is that *ttyname()* must be passed a file 9  
 1847 descriptor and returns the pathname of the terminal associated with that file descriptor, 9  
 1848 while *ctermid()* returns a string (such as */dev/tty*) that will refer to the controlling 9  
 1849 terminal if used as a pathname. Thus *ttyname()* is useful only if the process already has 9  
 1850 at least one file open to a terminal. 9

1851 **B.4.7.1 Generate Terminal Pathname**

1852 *L\_ctermid* must be defined appropriately for a given implementation and must be greater c  
 1853 than zero so that array declarations using it are accepted by the compiler. The value c  
 1854 includes the terminating null byte. c

1855 **B.4.7.2 Determine Terminal Device Name**

1856 The term “terminal” is used instead of the historical term “terminal device” in order to  
 1857 avoid a reference to an undefined term.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

#### 1858 B.4.8 Configurable System Variables B

1859 This section was added in response to requirements of application developers, and B  
 1860 particularly the X/OPEN system vendors. It is closely related to Configurable Pathname B  
 1861 Variables §B.5.7 as well. B

1862 Although a portable application can run on all systems by never demanding more B  
 1863 resources than the minimum values published in the standard, it is useful for that B  
 1864 application to be able to use the actual value for the quantity of a resource available on B  
 1865 any given system. To do this, the application will make use of the value of a symbolic B  
 1866 constant in <limits.h> or <unistd.h>. B

1867 However, once compiled, the application must still be able to cope if the amount of B  
 1868 resource available is increased. To that end, an application may need a means of B  
 1869 determining the quantity of a resource, or the presence of an option, at execution time. B

1870 Two examples are offered: B

1871 Applications may wish to act differently on systems with or without the Job B  
 1872 Control Option. Applications vendors who wish to distribute only a single binary B  
 1873 package to all instances of a computer architecture would be forced to assume job B  
 1874 control is never available if it were to rely solely on the <unistd.h> value B  
 1875 published in the standard. B

1876 International applications vendors occasionally require knowledge of the B  
 1877 {CLK\_TCK} value. Without the facilities of this section, they would be required B  
 1878 to either distribute their applications partially in source form or to have 50 Hertz B  
 1879 and 60 Hertz versions for the various countries they do business in. B

1880 It is the understanding that many applications are actually distributed widely in B  
 1881 executable form that lead to this facility. If limited to the most restrictive values in the B  
 1882 headers, such applications would have to be prepared to accept the most limited B  
 1883 environments offered by the smallest microcomputers. Although this is entirely portable, B  
 1884 it was felt by the Working Group that they should be able to take advantage of the B  
 1885 facilities offered by large systems, without the restrictions associated with source and B  
 1886 object distributions. B

1887 During the very heated arguments that accompanied the discussions of this feature, it was B  
 1888 pointed out that it is almost always possible for an application to discern what a value B  
 1889 might be at runtime by suitably testing the waters. And, in any event, it could always be B  
 1890 written to adequately deal with error returns from the various functions. In the end, it B  
 1891 was felt that this imposed an unreasonable level of complication and sophistication on B  
 1892 the application writer. B

1893 This runtime facility is not meant to provide ever-changing values that applications will B  
 1894 have to check multiple times. The values are seen as changing no more frequently than B  
 1895 once per system initialization, such as by a system administrator or operator with an B  
 1896 automatic configuration program. The standard specifies that they shall not change B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.  
 Do not specify or claim conformance to this document.



- 1897 within the lifetime of the process. B
- 1898 Some values apply to the system overall and others vary at the file system or directory B  
1899 level. These latter are described in Configurable Pathname Variables §B.5.7. B
- 1900 **B.4.8.1 Get Configurable System Variables** B
- 1901 Note that all values returned must be expressible as integers. The Working Group B  
1902 considered using string values, but the additional flexibility of this approach was rejected B  
1903 due to its added complexity of implementation and use. B
- 1904 Some values, such as {PATH\_MAX}, are sometimes so large that they must not be used B  
1905 to, say, allocate arrays. The *sysconf()* function will return a negative value to show that B  
1906 this symbolic isn't even defined, in this case. B
- 1907 **B.5 Files and Directories**
- 1908 See *pathname* resolution §2.4.
- 1909 **B.5.1 Directories**
- 1910 Historical implementations prior to 4.2BSD had no special functions, types, or headers  
1911 for directory access. Instead, directories were read with *read()* §6.4.1 and each program  
1912 that did so had code to understand the internal format of directory files. Many such  
1913 programs did not correctly handle the case of a maximum-length (historically fourteen  
1914 character) filename and would neglect to add a null character string terminator when  
1915 doing comparisons. The access methods in the standard eliminate that bug, as well as  
1916 hiding differences in implementations of directories or file systems.
- 1917 The directory access functions as described in an Appendix of the POSIX Trial Use  
1918 Standard were derived from 4.2BSD, were adopted in System V Release 3 and are in  
1919 *SVID* Volume 3, with the exception of a type difference for the *d\_ino* field. That field  
1920 represents implementation-dependent or even file system-dependent information (the i-  
1921 node number in most implementations). Since the directory access mechanism is  
1922 intended to be implementation independent, and since only system programs, not  
1923 ordinary applications, need to know about the i-node number (or file serial number §2.3)  
1924 in this context, the *d\_ino* field does not appear in the present standard. Also, programs  
1925 that want this information can get it with *stat()* §5.6.2.
- 1926 **B.5.1.1 Format of Directory Entries**
- 1927 Information similar to that in the header `<dirent.h>` is contained in a file `<sys/dir.h>` in  
1928 4.2BSD and 4.3BSD. The equivalent in these implementations of *struct dirent* from the  
1929 standard is *struct direct*. The filename was changed because the name `<sys/dir.h>` was c  
1930 also used in earlier implementations to refer to definitions related to the older access c  
1931 method; this produced name conflicts. The name of the structure was changed because c  
1932 the standard does not completely define what is in the structure, so it could be different c  
1933 on some implementations from *struct direct*. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.



- 1934 The name of a character array of an unspecified size should not be used as an *lvalue*. B  
 1935 Use of B
- 1936                    **sizeof (d\_name)** A
- 1937 is incorrect; use B
- 1938                    **strlen (d\_name)** A
- 1939 instead. B
- 1940 This description of the *d\_name* element was changed because the previous version gave B  
 1941 the impression that the character array *d\_name* was of a fixed size. Implementations may B  
 1942 need to declare *struct dirent* with an array size for *d\_name* of 1, but the actual number of B  
 1943 characters provided matches (or only slightly exceeds) the length of the file name. B
- 1944 Currently, implementations are excluded if they have *d\_name* with type *char \**. Lacking B  
 1945 experience of such implementations, the Working Group declined to try to describe in B  
 1946 standards language what to do if either type were permitted. B
- 1947 **B.5.1.2 Directory Operations**
- 1948 The returned value of *readdir()* merely *represents* a directory entry. No equivalence  
 1949 should be inferred.
- 1950 Since *readdir()* returns NULL both
- 1951    1. when it detects an error and
- 1952    2. when the end of the directory is encountered,
- 1953 an application that needs to tell the difference must set *errno* to zero before the call and  
 1954 check it if NULL is returned. Because the function must not change *errno* in case 2 and  
 1955 must set it to a non-zero value in case 1, zero *errno* after a call returning NULL indicates  
 1956 end of directory, otherwise an error:
- 1957 Routines to deal with this problem more directly were proposed.
- 1958                    **int derror (dirp)** A  
 1959                    **DIR \*dirp;** A
- 1960                    **void clearerr (dirp)** A  
 1961                    **DIR \*dirp;** A
- 1962 The first would indicate whether an error had occurred, and the second would clear the  
 1963 error indication. The simpler method involving *errno* was adopted instead by requiring  
 1964 that *readdir()* not change *errno* when end of directory is encountered.
- 1965 Historical implementations include two more functions.

|      |                                                                                                                                |   |
|------|--------------------------------------------------------------------------------------------------------------------------------|---|
| 1966 | <code>long telldir (dirp)</code>                                                                                               | A |
| 1967 | <code>DIR *dirp;</code>                                                                                                        | A |
| 1968 | <code>void seekdir (dirp, loc)</code>                                                                                          | A |
| 1969 | <code>DIR *dirp;</code>                                                                                                        | A |
| 1970 | <code>long loc;</code>                                                                                                         | A |
| 1971 | The <code>telldir()</code> function returns the current location associated with the named directory                           |   |
| 1972 | stream.                                                                                                                        |   |
| 1973 | The <code>seekdir()</code> function sets the position of the next <code>readdir()</code> operation on the directory            |   |
| 1974 | stream. The new position reverts to the one associated with the directory stream when                                          |   |
| 1975 | the <code>telldir()</code> operation was performed.                                                                            | B |
| 1976 | These functions have restrictions on their use related to implementation details. Their                                        |   |
| 1977 | capability can also be accomplished by saving a filename found by <code>readdir()</code> and later                             |   |
| 1978 | using <code>rewinddir()</code> and a loop on <code>readdir()</code> to relocate the position from which the                    |   |
| 1979 | filename was saved. Though this method is probably slower than using <code>seekdir()</code> and                                |   |
| 1980 | <code>telldir()</code> , there are few applications in which the capability is needed. For these reasons,                      |   |
| 1981 | the Working Group decided not to include <code>seekdir()</code> and <code>telldir()</code> in the standard.                    |   |
| 1982 | An error or signal indicating that a directory has changed while open was considered but                                       | A |
| 1983 | rejected.                                                                                                                      | A |
| 1984 | <b>B.5.2 Working Directory</b>                                                                                                 |   |
| 1985 | <b>B.5.2.1 Change Current Working Directory</b>                                                                                |   |
| 1986 | <b>B.5.2.2 Working Directory Pathname</b>                                                                                      |   |
| 1987 | Since the maximum pathname length is arbitrary unless <code>{PATH_MAX}</code> is defined, an                                   | B |
| 1988 | application cannot supply a <i>buf</i> with <i>size</i> <code>{{PATH_MAX} + 1}</code> in general.                              | B |
| 1989 | Having the routine take no arguments and instead use the C function <code>malloc()</code> to produce                           |   |
| 1990 | space for the returned argument was considered. The advantage is that <code>getcwd()</code> knows                              |   |
| 1991 | how big the working directory pathname is and can allocate an appropriate amount of                                            |   |
| 1992 | space. But the programmer would have to use the C function <code>free()</code> to free the resulting                           |   |
| 1993 | object, or each use of <code>getcwd()</code> would further reduce the available address space. Also,                           |   |
| 1994 | <code>malloc()</code> and <code>free()</code> are used nowhere else in the present standard. Finally, <code>getcwd()</code> is |   |
| 1995 | taken from the <i>SVID</i> , where it has the two arguments used in the standard.                                              |   |
| 1996 | The older function <code>getwd()</code> was rejected for use in this context because it had only a                             |   |
| 1997 | buffer argument and no <i>size</i> argument, and thus had no way to prevent overwriting the                                    |   |
| 1998 | buffer, except to depend on the programmer to provide a large enough buffer.                                                   |   |
| 1999 | The result if a NULL argument is passed to <code>getcwd()</code> is left implementation defined                                | A |
| 2000 | because some implementations dynamically allocate space in that case.                                                          | A |

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.



2001 If a program is operating in a directory where some (grand)parent directory does not  
 2002 permit reading, *getcwd()* may fail, as in most implementations it must read the directory  
 2003 to determine the name of the file. This can occur if search but not read permission is  
 2004 granted in an intermediate directory, or if the program is placed in that directory by some  
 2005 more privileged process (e.g. *login*). Including this error makes the reporting of the  
 2006 error consistent, and warns the application writer that *getcwd()* can fail for reasons  
 2007 beyond his control. (The other two failures should not be beyond his control.) Some  
 2008 implementations can avoid this occurrence (e.g. by implementing *getcwd()* using *pwd()*,  
 2009 and making *pwd()* a set-user-root process), thus the error was made optional.

2010 Because the standard permits the addition of other errors, this would be a common  
 2011 addition and yet one that applications could not be expected to deal with without this  
 2012 addition.

### 2013 B.5.3 General File Creation

#### 2014 B.5.3.1 Open a File

2015 Except as specified in the standard, the flags allowed in *oflag* are not mutually exclusive  
 2016 and any number of them may be used simultaneously.

2017 See *getgroups* §B.4.2.3 about the group of a newly-created file.

2018 The use of *open()* §5.3.1 to create a regular file is preferable to the use of *creat()* §5.3.2  
 2019 because the latter is redundant and included only for historical reasons.

2020 Implementations may deny access and return [EACCES] for reasons other than just those  
 2021 listed in the [EACCES] definition.

#### 2022 B.5.3.2 Create a New File or Rewrite an Existing One

2023 This interface is redundant. Its services are also provided by the *open()* function. It has  
 2024 been included primarily for historical purposes since many existing applications depend  
 2025 on it.

#### 2026 B.5.3.3 Set File Creation Mask

2027 Unsigned argument and return types for *umask()* were proposed. The return type was  
 2028 left unchanged, but the argument was changed to *mode\_t* §B.2.6.

#### 2029 B.5.3.4 Link to a File

2030 See directory entry §B.2.3.

2031 Linking to a directory is restricted to the super-user in most historical implementations  
 2032 because this capability may produce loops in the file hierarchy or otherwise corrupt the  
 2033 file system. However, file system implementations may be envisioned where multiple  
 2034 parents of a directory are handled without adverse side effects. Therefore, the standard  
 2035 does not require the restriction to the super-user. But see *rename()* §B.5.5.3. See also  
 2036 *unlink()* §5.5.1.



2037 **B.5.4 Special File Creation**2038 **B.5.4.1 Make a Directory**2039 See *mode\_t* §B.2.6.

2040 This function originated in 4.2BSD and was added to System V in Release 3.0, following  
2041 the Trial Use Standard.

2042 4.3BSD detects [ENAMETOOLONG].

2043 See *getgroups* §B.4.2.3 about the group of a newly-created directory.2044 **B.5.4.2 Make a FIFO Special File**

2045 The syntax of this routine is intended to maintain compatibility with existing 9  
2046 implementations of *mknod()*. The latter function was included in the 1984 *usr/group* A  
2047 *Standard*, but only for use in creating FIFO special files. The *mknod()* function was A  
2048 excluded from POSIX as implementation defined and replaced by *mkdir()* §5.4.1 and A  
2049 *mkfifo()* §5.4.2. A

2050 See *getgroups* §B.4.2.3 about the group of a newly-created FIFO.2051 **B.5.5 File Removal**

2052 Although *rmdir()* and *rename()* originated in 4.2BSD, the behavior specified for when  
2053 the directory to be removed does not exist or *new* already exists (returning [EEXIST] in  
2054 *errno*) is not compatible with 4.2BSD or 4.3BSD, which return [ENOTEMPTY]. B  
2055 Therefore, either value is allowed by the standard. The function was added to System V  
2056 in Release 3.0 but uses [ENOENT] where the standard uses [ENAMETOOLONG]. B  
2057 Volume 3 of the *SVID*, page 129, states: “FUTURE DIRECTION: To conform with the B  
2058 IEEE POSIX standard, when it is adopted as a full-use standard, the value of *errno* B  
2059 indicating that ...” B

2060 The Berkeley implementations of *rmdir()* and *rename()* used [ENOTEMPTY] for this B  
2061 error condition. When the *usr/group* Standard was published, it contained [EEXIST] B  
2062 instead. When AT&T adopted these functions into System V, they used the *usr/group* B  
2063 Standard as their reference. Therefore, several existing applications and implementations B  
2064 support/use both forms and the Working Group could not agree on either value. All B  
2065 implementations are required to supply both [EEXIST] and [ENOTEMPTY] in `<errno.h>` B  
2066 with distinct values so that applications can use both values in C language case B  
2067 statements. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

2068 **B.5.5.1 Remove Directory Entries**

2069 Unlinking a directory is restricted to the super-user in many historical implementations  
2070 for reasons given in *link()* §B.5.3.4. But see *rename()* §B.5.5.3.

2071 **B.5.5.2 Remove a Directory**

2072 See also [ENOTEMPTY] and [ENAMETOOLONG] §B.5.5. B

2073 **B.5.5.3 Rename a File**

2074 This *rename()* call is equivalent for regular files to that defined by the C Standard. Its  
2075 inclusion here expands that definition to include actions on directories and specifies  
2076 behavior when the *new* parameter names a file that already exists. That specification  
2077 requires that the action of the function be atomic.

2078 One of the reasons for introducing this function was to have a means of renaming  
2079 directories while permitting implementations to prohibit the use of *link()* §5.3.4 and  
2080 *unlink()* §5.5.1 with directories, thus constraining links to directories to those made by c  
2081 *mkdir()* §5.4.1. c

2082 The specification that if *old* and *new* refer to the same file describes existing, although c  
2083 undocumented, 4.3BSD behavior. It is intended to guarantee that: c

2084 `rename("x", "x");` A

2085 does not remove the file. c

2086 Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

2087 See also [[ENOTEMPTY] and [ENAMETOOLONG] §B.5.5. B

2088 **B.5.6 File Characteristics**

2089 The function *ustar()*, which appeared in the 1984 *usr/group Standard* and is still in the  
2090 *SVID*, was removed from the present standard before Trial Use because it was:

2091 • Not reliable. The amount of space available can change between the time the call is  
2092 made and the time the calling process attempts to use it.

2093 • Not required. The only known program that uses it is the text editor *ed*.

2094 It was also not readily extensible to networked systems.

2095 **B.5.6.1 File Characteristics: Header File and Data Structure**

2096 See *dev\_t* §B.2.6, *link\_t* §B.2.6, *mode\_t* §B.2.6, *off\_t* §B.2.6, and *uid\_t* §B.2.6.

2097 The S\_ISUID and S\_ISGID bits may be cleared on any write, not just on *open()* §5.3.1, as B  
2098 some historical implementations do it.

2099 System calls that update the time entry fields in the *stat* structure must be documented by c  
2100 the implementors. It is not expected that routines that call one of these system calls need c  
2101 to document this as a side effect. (Note that this includes most of the *stdio* routines in the c  
2102 *ANSI/X3.159-198x Programming Language C Standard*.) POSIX conforming systems c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

2103 should not update the time entry fields for functions listed in the standard unless the c  
2104 standard requires that they do, except in the case of documented extensions to the c  
2105 standard. c

#### 2106 B.5.6.2 Get File Status

2107 The intent of the paragraph describing “additional implementation defined access c  
2108 constraints” is to allow a secure implementation where a process with a label that does c  
2109 not dominate the file’s label cannot perform a *stat()* function. This is not related to read c  
2110 permission; a process with a label that dominates the file’s label will not need read c  
2111 permission. An implementation that supports write-up operations could fail *fstat()* c  
2112 function calls even though it has a valid file descriptor open for writing. c

#### 2113 B.5.6.3 File Accessibility

2114 Some Working Group discussions centered around inadequacies in the *access()* function B  
2115 led to the creation of an *eaccess()* function because: B

- 2116 1. Historical implementations of *access()* don’t test file access correctly when the B  
2117 process’s real user ID is super-user. In particular, they always return zero when B  
2118 testing execute permissions without regard to whether the file is executable. B
- 2119 2. The super-user has complete access to all files on a system. As a consequence, B  
2120 programs started by the super-user and switched to the effective user ID with lesser B  
2121 privileges cannot use *access()* to test their file access permissions. B

2122 After *eaccess()* was reviewed, the Working Group found that it still didn’t resolve B  
2123 problem 1, so the standard now allows *access()* to behave in the desired way because B  
2124 several implementations have corrected the problem. It was also argued that problem 2 B  
2125 is more easily solved by using *open()*, *chdir()*, or *exec()* functions as appropriate and B  
2126 responding to the error there, rather than creating a new function that wouldn’t be as B  
2127 reliable. Therefore, *eaccess()* was taken back out of the standard. B

2128 Secure implementations will probably need an extended *access()*-like function, but the B  
2129 Working Group did not have enough of the requirements to define it yet. This could be B  
2130 proposed as an extension to the Full Use standard. B

2131 The phrase “an implementation may substitute search permissions for execute c  
2132 permission” reflects the two possibilities implemented by historical implementations c  
2133 when checking super-user access for X\_OK. c

#### 2134 B.5.6.4 Change File Modes



### 2135 B.5.6.5 Change Owner and Group of File

2136 System III and System V allow a user to give away files, that is, the owner of a file may  
 2137 change its user ID to anything. This is a serious problem for implementations which are  
 2138 intended to meet government security regulations. Version 7 and 4.3BSD permit only the  
 2139 super-user to change the user ID of a file. Some government agencies (usually not ones  
 2140 concerned directly with security) find this limitation too confining. The standard uses  
 2141 “may” to permit secure implementations while not disallowing System V.

2142 System III and System V allow the owner of a file to change the group ID to anything.  
 2143 Version 7 permits only the super-user to change the group ID of a file. 4.3BSD permits  
 2144 the owner to change the group ID of a file to its effective group ID or to any of the groups  
 2145 in the list of supplementary group IDs, but to no others.

2146 The decision to require that, for non-privileged processes, the S\_ISUID and S\_ISGID bits  
 2147 be cleared on regular files but only *may* be cleared on non-regular files was to allow plans  
 2148 for using these bits in implementation specified manners on directories. Similar cases  
 2149 could be made for other file types, so the standard does not require that these bits be  
 2150 cleared except on regular files. Note that as these cases arise, the system implementors  
 2151 will have to determine whether these features enable any security loopholes and specify  
 2152 appropriate restrictions.

### 2153 B.5.6.6 Set File Access and Modification Times

2154 The *actime* structure member must be present, so that an application may set it, even  
 2155 though an interface implementation may ignore it and not change the access time on the  
 2156 file. If an application intends to leave one of the times of a file unchanged while  
 2157 changing the other, it should use *stat()* §5.6.2 to retrieve the file’s *st\_atime* §5.6.1.2.2  
 2158 and *st\_mtime* §5.6.1.2.2 parameters, set *actime* and *modtime* in the buffer, and change  
 2159 one of them before making the *utime()* call.

### 2160 B.5.7 Configurable Pathname Variables

2161 When the runtime facility described in Configurable Pathname Variables §B.4.8 was  
 2162 designed, it was realized that some variables change depending on the file system. For  
 2163 example, it is quite feasible for a system to have two varieties of file systems mounted:  
 2164 System V, and; a Berkeley “Fast File System.”

2165 If limited to strictly compile-time features, no application that was widely distributed in  
 2166 executable binary form could rely on more than 14 bytes in a pathname component, as  
 2167 that is the minimum published for {NAME\_MAX} in this standard. The *pathconf()*  
 2168 function allows the application to take advantage of the most liberal file system available  
 2169 at runtime. In many Berkeley-based systems, 255 bytes are allowed for pathname  
 2170 components.

2171 These values are potentially changeable at the directory level, not just at the file system.  
 2172 And, unlike the overall system variables, there is no guarantee that these might not  
 2173 change during program execution. However, if the program is dealing with an open file  
 2174 descriptor, using the *fpathconf()* function, they won’t change while the file is still open.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

**2175 B.5.7.1 Get Configurable Pathname Variables**

2176 The *pathconf()* function was proposed immediately after the *sysconf()* function when it c  
2177 was realized that some configurable values may differ across file system, directory, or c  
2178 device boundaries. c

2179 For example, {NAME\_MAX} frequently changes between System V and BSD-based file c  
2180 systems; System V uses a maximum of 14, Berkeley 255. On an implementation that c  
2181 provided both types of file systems, an application would be forced to limit all pathname c  
2182 components to 14 bytes, as this would be the value specified in <limits.h> on such a c  
2183 system. c

2184 Therefore, various useful values can be queried on any pathname or file descriptor, c  
2185 assuming that the appropriate permissions are in place. c

2186 Note that, unlike the values returned by *sysconf()*, the pathname-oriented variables are c  
2187 potentially more volatile and are not guaranteed to remain constant throughout the c  
2188 process's lifetime. For example, in between two calls to *pathconf()* the file system in c  
2189 question may have been unmounted and remounted with different characteristics. c

**2190 B.6 Input and Output Primitives**

2191 Rationale for the Change from O\_NDELAY to O\_NONBLOCK.

2192 System III and System V have included a flag, O\_NDELAY, to mark file descriptors so  
2193 that user processes would not block when doing I/O to them. If the flag is set, a *read()*  
2194 §6.4.1 or *write()* §6.4.2 call which would otherwise need to block for data returns a value  
2195 of zero instead. But a *read()* call also returns a value of zero on end of file, and  
2196 applications have no way to distinguish between these two conditions.

2197 BSD systems support a similar feature through a flag with the same name, but somewhat  
2198 different semantics. The flag applies to all users of a file (or socket) rather than only to  
2199 those sharing a file descriptor. The BSD interface provides a solution to the problem of  
2200 distinguishing between a blocking condition and an end of file condition by returning an  
2201 error, [EWOULDBLOCK], on a blocking condition.

2202 The 1984 *usr/group Standard* includes an interface with some features from both AT&T  
2203 and BSD. The overall semantics are that it applies only to a file descriptor. However, the  
2204 return indication for a blocking condition is an error, [EAGAIN]. This was the starting  
2205 point for POSIX.

2206 The problem with the 1984 *usr/group Standard* that it does not allow compatibility with  
2207 existing applications. An implementation cannot both conform to this standard and  
2208 support applications written for existing AT&T or BSD systems. This was the cause of at  
2209 least one objection during the trial-use balloting. Several changes have been considered,  
2210 either at that time or more recently, to address this issue. These include:

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.



- 2211 0) no change (from 1984 *usr/group Standard*)  
 2212 1) changing to System III/V semantics  
 2213 2) changing to BSD semantics  
 2214 3) broadening the standard to allow conforming implementation a choice  
 2215 among these semantics  
 2216 4) changing the name of the flag from O\_NDELAY  
 2217 5) changing to System III/V semantics and providing a new call to distinguish  
 2218 between blocking and end of file conditions

2219 The consensus of the Working Group at the January, 1986, meeting in Denver, was that c  
 2220 (4) is the best alternative. The new name is O\_NONBLOCK. This alternative allows a  
 2221 conforming implementation to provide backward compatibility at the source and/or  
 2222 object level with either AT&T or BSD systems (but the standard does not require or even  
 2223 suggest that this be done). It also allows Conforming Application Using Extensions the  
 2224 functionality to distinguish between blocking and end of file conditions, and to do so in  
 2225 as simple a manner as any of the alternatives. The greatest shortcoming was that it forces  
 2226 all existing AT&T and BSD applications that use this facility to be modified in order to c  
 2227 strictly conform to the standard. This same shortcoming applies to (0) and (3) as well,  
 2228 and it applies to one group of applications for (1), (2), and (5).

2229 Systems may choose to implement both O\_NDELAY and O\_NONBLOCK, and there is no  
 2230 conflict as long as an application does not turn both flags on at the same time.

2231 See also scope §B.6.5.1.

### 2232 B.6.1 Pipes

2233 The requirement that attempts to write on *fildev*[0] or to read on *fildev*[1] shall fail does  
 2234 not make the 4.3BSD implementation of pipes as sockets nonconforming, since the pipe  
 2235 code carefully sets up a pair of unidirectional sockets. System V Release 3 as distributed  
 2236 does not use streams for pipes. The historical (Version 7) error for such an attempt is  
 2237 [EBADF]

#### 2238 B.6.1.1 Create an Inter-Process Channel

2239 The wording carefully avoids using the verb “to open” in order to avoid any implication  
 2240 of use of *open()* §5.3.1.

2241 See also Write to a Pipe §B.6.4.2.



## 2242 B.6.2 File Descriptor Manipulation

### 2243 B.6.2.1 Duplicate an Open File Descriptor

2244 These interfaces are redundant. Their services are also provided by the *fcntl()* function. 9  
2245 They have been included in this standard primarily for historical reasons, since many 9  
2246 existing applications use them. 9

2247 In the description of [EBADF] the case of *fdes* being out of range is covered by the  
2248 given case of *fdes* not being valid. The descriptions for *fdes* and *fdes2* are different  
2249 because the only kind of invalidity that is relevant for *fdes2* is whether it is out of range,  
2250 that is, it does not matter whether *fdes2* refers to an open file when the *dup2()* call is  
2251 made.

2252 If *fdes2* is a valid file descriptor, it shall be closed, regardless of whether the function c  
2253 returns an indication of success or failure, unless *fdes2* is equal to *fdes*. c

## 2254 B.6.3 File Descriptor Deassignment

### 2255 B.6.3.1 Close a File

2256 Once a file is closed, the file descriptor no longer exists, since the integer corresponding  
2257 to it no longer refers to a file.

## 2258 B.6.4 Input and Output

2259 The standard permits return of the number of bytes read or written after an interrupted  
2260 operation in order to promote compatibility with System V, even though it makes writing  
2261 a Conforming Application more difficult.

2262 Whether the return values of, and *nbyte* arguments to, *read()* §6.4.1 and *write()* §6.4.2  
2263 should be signed or unsigned was a chronic source of controversy. On machines where  
2264 type *int* is of sixteen bits, only 32767 bytes may be transferred on one function call. If  
2265 *nbyte* were unsigned, it would be convenient for the return value to be of the same type.  
2266 But if the returned value were unsigned, it would be necessary to compare it to  
2267 (unsigned)-1 in order to detect an error. Although a definition such as *IO\_ERR* could be  
2268 provided to simplify code, still many existing applications would not conform.

2269 The Working Group decided to make *nbyte* unsigned, with the results of use of values  
2270 greater than {*INT\_MAX*} (often 32767) being made implementation defined. However,  
2271 the return value was left signed to avoid the error-detection problem. It is still possible to  
2272 compare the return value directly with *nbyte*, since the C Standard specifies that the  
2273 comparison will be done unsigned.

2274 Use of the type *long* was considered in order to avoid the sixteen bit problem, but not  
2275 adopted.

2276 New functions like *read()* and *write()* called *lread()* and *lwrite()* and differing only in  
2277 that their *nbyte* argument and return values would be of type *off\_t* §2.8 were proposed but  
2278 rejected. The Working Group is not necessarily against the creation of *lread()* and

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

2279 *lwrite()* calls, but was unable to clearly identify the need given the above. It was also  
 2280 noted that C has similar constraints parallel to those mentioned above, and that the type  
 2281 of *sizeof* is not necessarily long (where the largest object cannot exceed  
 2282 *sizeof(char[MAXINT])*).

2283 There were recommendations to add format parameters to *read()* and *write()* in order to A  
 2284 handle networked transfers among heterogenous file system and base hardware types. c  
 2285 Such a facility may be required for support by the OSI presentation of layer services. c  
 2286 However, the Working Group determined that this should correspond with similar C c  
 2287 Language facilities, and that is beyond the scope of the 1003 effort. The concept was c  
 2288 suggested to X3J11 for their consideration as a possible area for future work. c

2289 In 4.3BSD, a signal does not interrupt a *read()* §6.4.1 or a *write()* §6.4.2; thus the notes  
 2290 below regarding *setjmp()* §8.3.1 and *longjmp()* §8.3.1. In 4.2BSD, 4.3BSD, and  
 2291 Version 8 there is an additional function, *select()* §B.6.4, whose purpose is to pause until  
 2292 specified activity (data to read, space to write, etc.) is detected on specified file  
 2293 descriptors. It is common in applications written for those systems for *select()* to be used  
 2294 before *read()* in situations (such as keyboard input) where interruption of I/O due to a  
 2295 signal is desired. But this approach does not conform, because *select()* is not in the  
 2296 standard. The Working Group included *setjmp()* and *longjmp()* so that there would be a  
 2297 method usable by Conforming Application Using Extensions. 4.3BSD semantics are  
 2298 permitted by not requiring the implementation to return [EINTR] on a *read()* or *write()*.

2299 The standard permits *read()* and *write()* to return the number of bytes successfully  
 2300 transferred when interrupted by an error. This is not required because it is incompatible  
 2301 with Version 7, System III, and System V.

#### 2302 B.6.4.1 Read from a File

2303 The file offset is not incremented if an error is returned. c

2304 B

2305 References to actions taken on an “unrecoverable error” have been removed. It is B  
 2306 considered beyond the scope of this standard to describe what happens in the case of B  
 2307 hardware errors. B

#### 2308 B.6.4.2 Write to a File

2309 An attempt to write to a pipe or FIFO has several major characteristics: c

##### 2310 Atomic/non-atomic c

2311 A write is atomic if the whole amount written in one operation is not interleaved c  
 2312 with data from any other process. This is useful when there are multiple writers c  
 2313 sending data to a single reader. Applications need to know how large a write c  
 2314 request can be expected to be performed atomically. We call this maximum c  
 2315 {PIPE\_BUF}. The standard does not say whether write requests for more than c  
 2316 {PIPE\_BUF} bytes will be atomic, but requires that writes of {PIPE\_BUF} or less c  
 2317 bytes shall be atomic. c



|      |                                                                                 |                                                                                            |   |
|------|---------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|---|
| 2318 | Blocking/immediate                                                              |                                                                                            | c |
| 2319 |                                                                                 | Blocking is only possible with <code>O_NONBLOCK</code> clear. If there is enough space for | c |
| 2320 |                                                                                 | all the data requested to be written immediately, the implementation should do so.         | c |
| 2321 |                                                                                 | Otherwise, the process may block, that is, pause until enough space is available for       | c |
| 2322 |                                                                                 | writing. The effective size of a pipe or FIFO (the maximum amount that can be              | c |
| 2323 |                                                                                 | written in one operation without blocking) may vary dynamically, depending on              | c |
| 2324 |                                                                                 | the implementation, so it is not possible to specify a fixed value for it.                 | c |
| 2325 | Complete/partial/deferred                                                       |                                                                                            | c |
| 2326 | A write request,                                                                |                                                                                            | c |
| 2327 |                                                                                 | <code>int fildes, nbyte, ret;</code>                                                       | A |
| 2328 |                                                                                 | <code>char *buf;</code>                                                                    | A |
| 2329 |                                                                                 | <code>ret = write(fildes, buf, nbyte);</code>                                              | A |
| 2330 | may return                                                                      |                                                                                            | c |
| 2331 | complete:                                                                       | <code>ret = nbyte</code>                                                                   | c |
| 2332 | partial:                                                                        | <code>ret &lt; nbyte</code>                                                                | c |
| 2333 |                                                                                 | This shall never happen if $nbyte \leq \{PIPE\_BUF\}$ . If it does                         | c |
| 2334 |                                                                                 | happen (with $nbyte > \{PIPE\_BUF\}$ ), the standard does not                              | c |
| 2335 |                                                                                 | guarantee atomicity, even if $ret \leq \{PIPE\_BUF\}$ , because                            | c |
| 2336 |                                                                                 | atomicity is guaranteed according to the amount <i>requested</i> ,                         | c |
| 2337 |                                                                                 | not the amount written.                                                                    | c |
| 2338 | deferred:                                                                       | <code>ret = -1, errno = [EAGAIN]</code>                                                    | c |
| 2339 |                                                                                 | This error indicates that a later request may succeed. It does                             | c |
| 2340 |                                                                                 | not indicate that it <i>shall</i> succeed, even if $nbyte \leq$                            | c |
| 2341 |                                                                                 | $\{PIPE\_BUF\}$ , because if no process reads from the pipe or                             | c |
| 2342 |                                                                                 | FIFO, the write will never succeed. An application could                                   | c |
| 2343 |                                                                                 | usefully count the number of times [EAGAIN] is caused by a                                 | c |
| 2344 |                                                                                 | particular value of $nbyte > \{PIPE\_BUF\}$ and perhaps do later                           | c |
| 2345 |                                                                                 | writes with a smaller value, on the assumption that the                                    | c |
| 2346 |                                                                                 | effective size of the pipe may have decreased.                                             | c |
| 2347 | Partial and deferred writes are only possible with <code>O_NONBLOCK</code> set. |                                                                                            | c |
| 2348 | The relations of these properties are best shown in tables.                     |                                                                                            | c |

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.



| Write to a Pipe or FIFO with O_NONBLOCK clear. |                                 |                                 |                                  |
|------------------------------------------------|---------------------------------|---------------------------------|----------------------------------|
| immediately writable:                          | none                            | some                            | <i>nbyte</i>                     |
| $nbyte \leq \{PIPE\_BUF\}$                     | atomic blocking<br><i>nbyte</i> | atomic blocking<br><i>nbyte</i> | atomic immediate<br><i>nbyte</i> |
| $nbyte > \{PIPE\_BUF\}$                        | blocking<br><i>nbyte</i>        | blocking<br><i>nbyte</i>        | blocking<br><i>nbyte</i>         |

2359 If the O\_NONBLOCK flag is clear, a write request shall block if the amount writable  
 2360 immediately is less than that requested. If the flag is set (by *fcntl()*), a write request shall  
 2361 never block.

| Write to a Pipe or FIFO with O_NONBLOCK set. |                 |                                      |                                    |
|----------------------------------------------|-----------------|--------------------------------------|------------------------------------|
| immediately writable:                        | none            | some                                 | <i>nbyte</i>                       |
| $nbyte \leq \{PIPE\_BUF\}$                   | -1,<br>[EAGAIN] | -1,<br>[EAGAIN]                      | atomic<br><i>nbyte</i>             |
| $nbyte > \{PIPE\_BUF\}$                      | -1,<br>[EAGAIN] | < <i>nbyte</i><br>or -1,<br>[EAGAIN] | $\leq nbyte$<br>or -1,<br>[EAGAIN] |

2372 There is no way provided for an application to determine whether the implementation  
 2373 will ever perform partial writes to a pipe or FIFO. Every application should be prepared  
 2374 to handle partial writes when O\_NONBLOCK is set and the requested amount is greater  
 2375 than {PIPE\_BUF}, just as every application should be prepared to handle partial writes on  
 2376 other kinds of file descriptors.

2377 Where the standard requires -1 returned and *errno* set to [EAGAIN], most historical  
 2378 implementations return 0 (with the O\_NDELAY flag set: that flag is the historical  
 2379 predecessor of O\_NONBLOCK, but is not itself in the standard). The error indications in  
 2380 the standard were chosen so that an application can distinguish these cases from end of  
 2381 file. While *write()* cannot receive an indication of end of file, *read()* can, and the  
 2382 Working Group chose to make the two functions have similar return values. Also, some  
 2383 existing systems (e.g., Version 8) permit a write of zero bytes to mean that the reader  
 2384 should get an end of file indication: for those systems, a return value of zero from *write*  
 2385 indicates a successful write of an end of file indication.

2386 The concept of a {PIPE\_MAX} limit (indicating the maximum number of bytes that can  
 2387 be written to a pipe in a single operation) was discussed by the Working Group. The  
 2388 Group decided this concept would unnecessarily limit application writing.

2389 See also O\_NONBLOCK §B.6.

- 2390 The file offset is not incremented if an error is returned. C
- 2391 The standard does not specify behavior of concurrent writes to a file from multiple A  
2392 processes. Applications should use some form of concurrency control. A
- 2393 B
- 2394 References to actions taken on an “unrecoverable error” have been removed. It is B  
2395 considered beyond the scope of this standard to describe what happens in the case of B  
2396 hardware errors. B
- 2397 **B.6.5 Control Operations on Files**
- 2398 **B.6.5.1 Data Definitions for File Control Operations**
- 2399 The main distinction between the file descriptor flags and the file status flags is scope.  
2400 The former apply to a single file descriptor only, while the latter apply to all file  
2401 descriptors that share a common open file description (by inheritance through *fork()* C  
2402 §3.1.1 or an *F\_FDUPFD* operation with *fcntl()* §6.5.2). Neither apply to file descriptors  
2403 that have different file pointers even if they refer to the same file (by separate *open()*  
2404 §5.3.1 calls). For *O\_NONBLOCK*, this scoping is like that of *O\_NDELAY* in System V  
2405 rather than in 4.3BSD, where the scoping for *O\_NDELAY* is different from all the other  
2406 flags accessed via the same commands.
- 2407 For example:
- ```
2408         fd1 = open (pathname, oflags);           A
2409         fd2 = dup (fd1);                         A
2410         fd3 = open (pathname, oflags);           A
```
- 2411 Does an *fcntl()* call on *fd1* also apply to *fd2* or *fd3* or to both? According to the standard,
2412 *F_SETFD* applies only to *fd1*, while *F_SETFL* applies to *fd1* and *fd2* but not to *fd3*. This
2413 is in agreement with all common historical implementations except for BSD with the
2414 *F_SETFL* command and the *O_NDELAY* flag (which would apply to *fd3* as well). Note
2415 that this does not force any incompatibilities in BSD implementations, because
2416 *O_NDELAY* is not in the standard. See also *O_NONBLOCK* §B.6.
- 2417 **B.6.5.2 File Control**
- 2418 The ellipsis in the Synopsis is the syntax specified by the C Standard for a variable
2419 number of arguments. It is used because System V uses pointers for the implementation
2420 of file locking functions. A
2421 B
- 2422 POSIX permits concurrent read and write access to file data using the *fcntl()* function; B
2423 this is a change from the /usr/group Standard and previous drafts, which included a B
2424 *lockf()* function. Without concurrency controls, this feature may not be fully utilized A
2425 without occasional loss of data. Since other mechanisms for creating critical regions, A
2426 such as semaphores, are not included, a file record locking mechanism was thought A
2427 appropriate. The *fcntl()* mechanism may be used to implement semaphores, although A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 2428 access is not first-in-first-out without extra application implementation effort. A
- 2429 Data losses occur in several ways. One is that read and write operations are not atomic, A
 2430 and as such a reader may get segments of new and old data if concurrently written by A
 2431 another process. Another occurs when several processes try to update the same record, A
 2432 without sequencing controls; several updates may occur in parallel and the last writer A
 2433 will “win.” Another case is a b-tree or other internal list-based database that is A
 2434 undergoing reorganization. Without exclusive use to the tree segment by the updating A
 2435 process, other reading processes chance getting lost in the database when the index A
 2436 blocks are split, condensed, inserted, or deleted. While *fcntl()* is useful for many A
 2437 applications, it is not intended to be overly general, and will not handle the b-tree A
 2438 example well. A
- 2439 This facility is only required for regular files, because it is not appropriate for many
 2440 devices such as terminals and network connections. However, if it is not supported on a
 2441 given device, the *fcntl()* function must return an error of [ENODEV] B
- 2442 Since *fcntl()* works with “any file descriptor associated with that file, however it is B
 2443 obtained,” the file descriptor may have been inherited through a *fork()* §3.1.1 or *exec* B
 2444 §3.1.2 operation and thus may affect a file that another process also has open.
- 2445 The use of the open file description to identify what to lock requires extra calls and C
 2446 presents problems if several processes are sharing a open file description but there are too A
 2447 many implementations of the existing mechanism for the standard to use different A
 2448 specifications. A
- 2449 But note that while a open file description may be shared through *fork()*, locks are not A
 2450 inherited through *fork()*. Yet locks may be inherited through *exec()*. A
- 2451 Shared read locks are not part of the design because no easy implementation was seen A
 2452 that would eliminate the race conditions and lockout that would occur in normal usage. A
- 2453 Since locking is performed with *fcntl()*, rather than *lockf()*, this specification prohibits B
 2454 use of locking on a file that is not open for writing. B
- 2455 Before successful return from a F_SETLK or F_SETLKW request, the previous lock type B
 2456 for each byte in the specified region shall be replaced by the new lock type. This can B
 2457 result in a previously locked region being split into smaller regions. If this would cause B
 2458 the number of regions being held by all processes in the system to exceed a system- B
 2459 imposed limit, the *fcntl()* function returns -1 with *errno* set to [ENOLCK]. B
- 2460 Mandatory locking was a major feature of the 1984 *usrl/group Standard*. For advisory A
 2461 file record locking to be effective, all processes that have access to a file must cooperate A
 2462 and use the advisory mechanism before doing I/O on the file. Enforcement-mode record A
 2463 locking is important when it cannot be assumed that all processes are cooperating. For A
 2464 example, if one user uses an editor to update a file at the same time that a second user A
 2465 executes another process that updates the same file, if only one of the two processes is A
 2466 using advisory locking, the processes are not cooperating. Enforcement mode record A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

- 2467 locking would protect against accidental collisions. A
- 2468 Secondly, advisory record locking requires a process using locking to bracket each I/O A
 2469 operation with lock (or test) and unlock operations. With enforcement mode file and A
 2470 record locking, a process can lock the file once and unlock when all I/O operations have A
 2471 been completed. Enforcement mode record locking provides a base that can be enhanced, A
 2472 for example, with shareable locks. That is, the mechanism could be enhanced to allow a A
 2473 process to lock a file so other processes could read it but none of them could write it. A
- 2474 Mandatory locks were omitted for several reasons. A
- 2475 1. Mandatory lock setting was done by multiplexing the setgid bit in most A
 2476 implementations; this was confusing, at best. A
 - 2477 2. Relationship to file truncation as supported in 4.2BSD was not well specified. A
 - 2478 3. Any publicly readable file could be locked by anyone. Many historical A
 2479 implementations keep the password database in a publicly-readable file. A
 2480 A malicious user could thus prohibit logins. Another possibility would be to hold A
 2481 open a long-distance telephone line. A
 - 2482 4. Some demand-paged historical implementations offer memory mapped files, and A
 2483 enforcement cannot be done on that type of file. A
- 2484 Since sleeping on a region is interrupted with any signal, *alarm()* §3.4.1 may be used to 9
 2485 provide a timeout facility in applications requiring it. This is useful in deadlock A
 2486 detection. Although the *fcntl()* implementation must provide deadlock detection A
 2487 between processes that are related by locked resources, it does not have to account for A
 2488 deadlocks caused by activities unrelated to *fcntl()* that have suspended a lock owner. A
- 2489 The *l_start* element of the *flock* structure and the *offset* argument of *lseek()* are, in some B
 2490 cases, taken as signed offsets from some position in a file, but the type of these objects is B
 2491 allowed to be unsigned. This apparent conflict is avoided by the C Standard's definitions B
 2492 of conversions from signed to unsigned and of arithmetic operations on unsigned types. B
 2493 If *U* is of type *off_t*, the expressions B
- $$2494 \quad U + ((\text{off_t}) (-i)) \quad A$$
- 2495 and B
- $$2496 \quad U - i \quad A$$
- 2497 will produce the same result, and, for example, B
- $$2498 \quad \text{lseek} (\text{fd}, (\text{off_t}) - 4, \text{SEEK_END}); \quad A$$
- 2499 is well defined. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

2500 B.6.5.3 Reposition Read/Write File Offset

2501 The C Standard includes the functions *fgetpos()* §B.6.5.3 and *fsetpos()* §B.6.5.3 which
2502 work on very large files by use of a special positioning type.

2503 Although *lseek()* may position the file offset beyond the end of the file, this function does c
2504 not itself extend the size of the file. While the only function in POSIX that may extend
2505 the size of the file is *write()* §6.4.2, several Standard C functions, such as *fwrite()*,
2506 *fprintf()*, etc., may do so (by causing calls on *write()*).

2507 An illegal file offset that would cause [EINVAL] to be returned may be both c
2508 implementation defined and device dependent (for example, memory may have few
2509 illegal values). A negative file offset may be legal for some devices in some c
2510 implementations.

2511 See *fcntl()* §B.6.5.2 for a explanation of the use of signed and unsigned offsets with B
2512 *lseek()*. B

2513 B.7 Device- and Class-Specific Functions

2514 This section has probably undergone more debate and revision than any other in the A
2515 standard. Numerous historical implementations were investigated, and at least four A
2516 major proposals were made. A

2517 There are several sources of the difficulties of this section: A

- 2518 • The basic Version 7 *ioctl()* mechanism is difficult to specify adequately, due to its A
2519 use of a third argument that varies in both size and type according to the second, A
2520 command, argument. A

- 2521 • System III introduced and System V continued *ioctl()* commands that are completely A
2522 different from those of Version 7. A

- 2523 • 4.2BSD and other Berkeley systems added to the basic Version 7 *ioctl()* command A
2524 set; some of these were for features such as job control that POSIX eventually A
2525 adopted. A

- 2526 • None of the basic historical implementations are adequate in an international A
2527 environment. This concern is not technically within the scope of POSIX, but the A
2528 Working Group did not want to supply unnecessary impediments to A
2529 internationalization. A

2530 The 1984 *lusr/group Standard* attempted to specify a portable mechanism that A
2531 application writers could use to get and set the modes of an asynchronous terminal. The A
2532 intention of that committee was to provide an interface that was neither implementation A
2533 specific nor hardware dependent. Initial proposals dealt with high level routines similar A
2534 to the *curses* library (available on most historical implementations). In such an A
2535 implementation, the user interface would consist of calls similar to: A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 2576 the information in System V, but attempts to avoid problems of case, speed, A
 2577 networks, and internationalization. A
- 2578 Specific *tc_*()* functions A
 2579 to replace each *ioctl()* function were finally incorporated into the standard, instead A
 2580 of any of the above-mentioned proposals. A
- 2581 The issue of modem control [Unknown Reference Type] § was excluded from POSIX on A
 2582 the grounds that: A
- 2583 1. it was concerned with setting and control of hardware timers, and A
 - 2584 2. the appropriate timers and settings vary widely internationally. A
 - 2585 3. Feedback from X/OPEN indicated that this facility was not consistent with c
 2586 European needs, and that specification of such a facility was not a requirement for c
 2587 portability from their “international perspective.” c
- 2588 **B.7.1 General Terminal Interface**
- 2589 Although the Working Group attempted to take into account needs of both interface A
 2590 implementors and application developers throughout the standard, more attention was A
 2591 paid to the needs of the latter in this section. This is because, while many aspects of the A
 2592 programming interface can be hidden from the user by the application developer, the A
 2593 terminal interface is usually a large part of the user interface. Although to some extent A
 2594 the application developer can build missing features or work around inappropriate ones, A
 2595 the difficulties of doing that are greater in the terminal interface than elsewhere. For A
 2596 example, efficiency prohibits the average program from interpreting every character A
 2597 passing through it in order to simulate character erase, line kill, etc. These functions A
 2598 should usually be done by the operating system, possibly at interrupt level. A
- 2599 The *tc_*()* functions were introduced as a way of avoiding the problems inherent in the
 2600 traditional *ioctl()* §B.7.1 function and in variants of it that were proposed. For example,
 2601 *tcgets()* is specified in place of the use of the TCGETS *ioctl()* command function. This
 2602 allows specification of all the arguments in a manner consistent with the C Standard,
 2603 unlike the varying third argument of *ioctl()*, which is sometimes a pointer (to any of
 2604 many different types) and sometimes an int.
- 2605 The advantages of this new method include:
- 2606 • It allows strict type checking.
 - 2607 • The direction of transfer of control data is explicit.
 - 2608 • Portable capabilities are clearly identified.
 - 2609 • The need for a general interface routine is avoided.
- 2610 The disadvantages include

- 2611 • No historical implementation uses the new method.
- 2612 • There are many small routines instead of one general-purpose one.
- 2613 • The historical parallel with *fcntl()* §6.5.2 is broken.

2614 B.7.1.1 Interface Characteristics

2615 B.7.1.1.1 Description

2616 B.7.1.1.2 Opening a Terminal Device File

2617 B.7.1.1.3 Process Groups

2618 B.7.1.1.4 The Controlling Terminal

2619 B.7.1.1.5 Job Access Control

2620 The foreground/background check performed by the terminal driver must be repeatedly
 2621 performed until the calling process moves into the foreground. That is, when the
 2622 terminal driver determines that the calling process is in the background and should
 2623 receive a job control signal, it sends the appropriate signal (SIGTTIN or SIGTTOU) to
 2624 every process in the process group of the calling process and then it allows the calling
 2625 process to immediately receive the signal. The latter is typically performed by blocking
 2626 the process so that the signal is immediately noticed. Note, however, that after the
 2627 process finishes receiving the signal and control is returned to the driver, the terminal
 2628 driver must reexecute the foreground/background check. The process may still be in the
 2629 background, either because it was continued in the background by a job control shell, or
 2630 because it caught the signal and did nothing.

2631 The terminal driver repeatedly performs the foreground/background checks whenever a
 2632 process is about to access the terminal. In the case of *write()* or the **Control Functions**
 2633 §7.2, the check is performed at the entry of the function. In the case of *read()*, the check
 2634 is performed not only at the entry of the function but also after blocking the process to
 2635 wait for input characters (if necessary). That is, once the driver has determined that the
 2636 process calling the *read()* function is in the foreground, it attempts to retrieve characters
 2637 from the input queue. If the queue is empty, it blocks the process waiting for characters.
 2638 When characters are available and control is returned to the driver, the terminal driver
 2639 must return to the repeated foreground/background check again. The process may have
 2640 moved from the foreground to the background while it was blocked waiting for input
 2641 characters.

2642 See also **job control** §B.3.3. A

2643 B.7.1.1.6 Input Processing and Reading Characters C

2644

2645 B.7.1.1.7 Canonical Mode Input Processing

2646 4.3BSD has a WERASE character that erases the last “word” typed (but not any A
 2647 preceding blanks or tabs). A word is defined as a sequence of non-blank characters, with A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 2648 tabs counted as blanks. Like ERASE, WERASE does not erase beyond the beginning of A
 2649 the line. This WERASE feature has not been specified in the standard because it is A
 2650 difficult to define in the international environment. It is only useful for languages where A
 2651 words are delimited by blanks. In some ideographic languages, such as Japanese and A
 2652 Chinese, words are not delimited at all. The WERASE character should presumably take A
 2653 one back to the beginning of a sentence in those cases: practically, this means it would A
 2654 not get much use for those languages. Thus WERASE would be needless overhead, and A
 2655 has been omitted as superfluous. A
- 2656 **B.7.1.1.8 Non-Canonical Mode Input Processing**
 2657 See `c_min` and `c_time` §B.7.1.2.2. A
- 2658 **B.7.1.1.9 Writing Characters and Output Processing**
- 2659 **B.7.1.1.10 Special Characters**
 2660 Discussion: The character values for INTR, QUIT, ERASE, KILL, EOF, and EOL, may be c
 2661 changed to suit individual tastes. C
- 2662 **B.7.1.1.11 Modem Disconnect**
- 2663 **B.7.1.1.12 Closing a Terminal Device File**
- 2664 **B.7.1.2 Settable Parameters**
- 2665 **B.7.1.2.1 Synopsis**
- 2666 **B.7.1.2.2 *termios* Structure**
 2667 C
- 2668 **B.7.1.2.3 Input Modes**
- 2669 **B.7.1.2.4 Output Modes**
- 2670 **B.7.1.2.5 Control Modes**
- 2671 **B.7.1.2.6 Local Modes**
 2672 Non-canonical mode is provided to allow fast bursts of input to be read efficiently while B
 2673 still allowing single character input. B
- 2674 **B.7.1.2.7 Special Control Characters**

2675 B.7.2 General Terminal Interface Control Functions**2676 B.7.2.1 Get and Set State****2677 B.7.2.2 Line Control Functions****2678 B.7.2.3 Get Distinguished Process Group ID**

2679 The *tcgetpgrp()* and *tcsetpgrp()* functions have identical functionality to the 4.2BSD
2680 *ioctl()* functions TIOCGPGRP and TIOCSPGRP except for additional security restrictions
2681 imposed on *tcsetpgrp()*. The 4.2BSD TIOCSPGRP function allows the caller to associate
2682 the terminal with any process group. This allows a user to generate signals from the
2683 keyboard that can be sent to any desired process while bypassing the security restrictions
2684 imposed by *kill()*. To address this, *tcsetpgrp()* imposes security restrictions similar to
2685 *kill()*; the difference is the addition of the saved process group ID. This was added to
2686 allow a job control shell to return its controlling terminal to its original process group
2687 (the one in effect when the job control shell was executed) regardless of whether the user
2688 ID security checks permit it. (Typically the saved process group of a process matches the
2689 process group of its parent; but this is not necessarily so.) A job control shell does this
2690 before returning control back to its parent when it is terminating or suspending itself.
2691 See also *jcsetpgrp()* §B.4.3.2. Note that 4.3BSD closed the 4.2BSD security problem
2692 somewhat; it looks for a process whose process ID and process group ID are both equal to
2693 the process group supplied to TIOCSPGRP and requires that this process be a descendant
2694 of the calling process or that user IDs match. However this still has problems since there
2695 may be processes which belong to the specified process group, but which are not the
2696 process group leader. This is actually a frequent occurrence since *cs/h* makes the first
2697 process in a pipeline be the process group leader and this process is usually the first to
2698 terminate. See also job control §B.3.3.

A

2699 B.7.2.4 Set Distinguished Process Group ID

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

1 B.8 C Language Library

2 When the *ANSI/X3.159-198x Programming Language C Standard* is adopted, it will be
 3 the basis for a C language binding to POSIX. In the interim, the routines in this chapter
 4 are left unstandardized, but are defined by common usage and traditional
 5 implementations. Common usage may also be derived by such historical publications as
 6 *The C Programming Language*, by Kernighan and Ritchie, listed in Bibliographic Notes
 7 §B.11.

8 The null set of supported languages is allowed.

9 B.8.1 Referenced C Language Routines

10 B.8.1.1 Extensions to *asctime()* Function

11 System V uses the TZ environment variable to set some information about time. It has
 12 the form (spaces inserted for clarity):

13 *std offset dst*

14 where the first three characters (*std*) are the name of the standard time zone, the digits
 15 which follow (*offset*) are the hours West of Greenwich (or, if preceded by “-”, East),
 16 and the next three characters (*dst*) are the name of the summer time zone. Both *std* and
 17 *offset* are required; if *dst* is missing, summer time does not apply.

18 Currently, the UNIX system *localtime* function translates a number of seconds since The
 19 Epoch §2.3 into a detailed breakdown of that time. This breakdown includes:

- 20 • Time of day: Hours, minutes, and seconds.
- 21 • Day of the month, month of the year, and the year.
- 22 • Day of the week and day of the year (Julian day).
- 23 • Whether or not summer (daylight saving) time is in effect.

24 It is first and last items that present a nasty problem: The time of the day depends on
 25 whether or not summer time is in effect. Whether or not summer time is in effect
 26 depends on the locale and date.

27 Currently the UNIX system has built into it only the United States federal law for the
 28 years 1970 to 1986. The U.S. law was changed for 1987 and subsequent years, so much
 29 UNIX system software is now “broken.” Actually, 4.2BSD includes time zone rules in a
 30 file that does take Europe and Australia into account. There are some errors and
 31 limitations with this method. And if the system is outside the United States, that same
 32 UNIX system software has always been broken.

33 The challenge is to fix the existing built-in rules for the new U.S. law and, in the process,
 34 extend *localtime* so that non-U.S. locales won't suffer from Yankee daylight saving time.
 35 Fixing the built-in rule is straightforward. Extending *localtime* is less so.

36 This proposal extends the existing TZ environment variable (which names the locale's c
37 time zone) to also include a rule for when to use standard time and when to use summer c
38 time. Southern hemisphere time zones are supported by allowing the first *rule date* c
39 (change to summer time) to be later in the year than the second *rule date* (change to c
40 standard time). c

41 The proposal accommodates the “floating day” rules (for example “last Sunday in c
42 October”) used in the U.S. and Canada (and the European Economic Community for the c
43 last several years). In theory, TZ only has to be set once and then never touched again c
44 unless the law is changed. c

45 Julian dates are proposed with two syntaxes, one zero based, the other one based. They c
46 are here for historical reasons. The one based counting (*J*) is used more commonly in c
47 Europe (and on calendars people may use for reference). The zero based counting (*n*) is c
48 used currently in some implementations and should be kept for historical reasons as well c
49 as being the only way to specify Leap day. c

50 It is expected the leading slash followed by some bytes as either the entire TZ string or as c
51 the *rule* will enable systems to have time zone information included in a file (as 4.2BSD c
52 systems currently do) or use the bytes as an index into a database. The implementors c
53 have the option as to how these bytes are interpreted. Allowing the implementors to c
54 specify either the entire time zone or the *rule* makes the proposal capable of describing c
55 the complete history for a multitude of locales. This proposal speculates that very few c
56 programs actually need to be historically accurate as long as the relative timing of two c
57 events is preserved. But, for the probably few programs that do desire such accuracy, the c
58 */bytes* method is provided. c

59 Summer time is governed by both locale and date. This proposal only handles the locale c
60 dependency. Using an implementation defined file format for either the entire TZ c
61 variable or to specify the *rules* for a particular time zone is allowed as a means by which c
62 both the locale and date dependency can be handled. c

63 Since current implementations do not examine TZ beyond the assumed end of *dst*, it is c
64 possible to literally extend TZ and break very little existing software. Since much of the c
65 software doesn't work anyway outside the U.S. time zones, minor changes to TZ (such as c
66 extending *offset* to be *hh:mm* — as long as the colon and minutes, *:mm*, are optional) c
67 will have little impact. c

68 B.8.1.2 Extensions to *setlocale()* Function c

69 Recently, the ANSI X3J11 subcommittee issued a draft proposal for the C Programming c
70 Language. In addition to many changes to the language, the proposal defines a collection c
71 of interfaces to support internationalization. One of the most significant aspects of these c
72 interfaces is a facility to set and query the *international environment*. The international c
73 environment is a repository of information that affects the behavior of certain c
74 functionality, namely: c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 75 • Character Handling C
- 76 • String Handling (i.e., collating) C
- 77 • Date/Time Formatting C
- 78 • Numeric Editing C

79 The *setlocale()* function provides the application developer with the ability to set all or
80 portions, called *categories*, of the international environment. These categories correspond
81 to the areas of functionality, mentioned above. The syntax for *setlocale* is the following: C

```
82           char        *setlocale (category, locale)        A
83           int         category;                            A
84           char        *locale                               A
```

85 Where *category* is the name of one of four categories, namely: C

```
86           LC_CTYPE                                         A
87           LC_COLLATE                                       A
88           LC_TIME                                           A
89           LC_NUMERIC                                       A
```

90 In addition, a special value, called *LC_ALL*, directs *setlocale()* to set all categories. C

91 The *locale* argument is a character string that points to a specific setting for the
92 international environment, or locale. There are three preset values for the locale
93 argument, namely: C

94 *C* Specifies the minimal environment for C translation. If *setlocale* is C
95 not invoked, the "C" locale is the default. C

96 "" Specifies an implementation-defined native environment. C

97 *NULL* Used to direct *setlocale()* to query the current international C
98 environment and return the name of the locale. C

99 This section describes the behavior of an implementation of *setlocale()* and its use of C
100 environment variables in controlling this behavior on POSIX-based systems. There are C
101 two primary uses of *setlocale()*: C

- 102 • Querying the international environment to find out what it is set to, C
- 103 • Setting the international environment, or *locale*, to a specific value. C

104 The following sub-sections will describe the behavior of *setlocale()* in these two areas. C
105 Since it is difficult to describe the behavior in words, examples will be used to illustrate C
106 the behavior of specific uses. C

107 To query the international environment, *setlocale()* is invoked with a specific category C
108 and the null pointer as the locale. The null pointer is a special directive to *setlocale()* C
109 that tells it to query rather than set the international environment. Below is the syntax for C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

110 using *setlocale()* to query the name of the international environment: c

111 *setlocale()* returns the string corresponding to the current international environment. c

112 This value may be used by a subsequent call to *setlocale()* to reset the international c

113 environment to this value. However, it should be noted that the return value from c

114 *setlocale()* is a pointer to a static area within the function and is not guaranteed to remain c

115 unchanged (i.e., it may be modified by a subsequent call to *setlocale()*). Therefore, if the c

116 purpose of calling *setlocale()* is to save the value of the current international c

117 environment so it can be changed and reset back later, the return value should be copied c

118 to a character array in the calling program. c

119 There are three ways to set the international environment with *setlocale()*: c

120 **setlocale(category, string)** c

121 This usage will set a specific *category* in the international c

122 environment to a specific value corresponding to the value of the c

123 *string*. A specific example is provided below: c

124 **setlocale(LC_ALL, "Fr_FR.8859");** A

125 In this example, all categories of the international environment c

126 will be set to the locale corresponding to the string c

127 "Fr_FR.8859", or the french language as spoken in France c

128 using the ISO 8859/1 code set. c

129 If the string does not correspond to a valid locale, *setlocale()* will c

130 return a null pointer and the international environment is not c

131 changed. Otherwise, *setlocale* will return the name of the locale c

132 just set. c

133 **setlocale(category, "C")** c

134 The ANSI X3J11 draft proposal states that one locale must exist c

135 on all conforming implementations. The name of the locale is c

136 "C", and corresponds to a minimal international environment c

137 needed to support the C programming language. c

138 **setlocale(category, "")** c

139 This will set a specific category to an implementation-defined c

140 default. For POSIX-based systems, this corresponds to the value of c

141 the environment variables. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

142 B.8.2 FILE-Type C Language Functions

143 B.8.2.1 Map a Stream Pointer to a File Descriptor

144 Without some specification of which file descriptors are associated with these streams, it
 145 is impossible for an application to set up the streams for another application it starts with
 146 *fork()* §3.1.1 and *exec* §3.1.2. In particular, it would not be possible to write a portable
 147 version of the *sh* command processor (although there may be other constraints that
 148 would prevent that portability).

149 Note that this standard permits an implementation to associate file descriptors other than
 150 0, 1, and 2 with *stdin*, *stdout*, and *stderr*.

151 B.8.2.2 Open a Stream on a File Descriptor

152 The file descriptor may have been obtained from *open()* §5.3.1, *creat()* §5.3.2, *pipe()*
 153 §6.1.1, *dup()* §6.2.1, *fcntl()* §6.5.2, or inherited through *fork()* §3.1.1 or *exec* §3.1.2, or
 154 perhaps obtained by implementation-dependent means, such as the 4.3BSD *socket()* call.

155 The meanings of the *type* arguments of *fdopen* and *fopen* differ. With *fdopen*, open for
 156 write (“w” or “w+”) does not truncate and append (“a” or “a+”) cannot create for
 157 writing. There is no need for “b” in the format due to the equivalence of binary and text
 158 files in POSIX. See Text vs. binary file modes §B.1.4.

159 B.8.3 Other C Language Functions

160 B.8.3.1 Non-Local Jumps

161 X3J11 specifies various restrictions on the usage of the *setjmp()* macro in order to
 162 permit implementors to recognize the name in the compiler and not implement an actual
 163 function. These same restrictions apply to the *sigsetjmp()* macro.

164 The names of these functions were changed to *sigsetjmp()* and *siglongjmp()*. This
 165 avoided conflict with the C Standard *setjmp()* and *longjmp()*, which do not have the
 166 same behavior in regards to signal masks.

167 There are processors that cannot easily support these calls, but the Working Group did
 168 not consider that a sufficient reason not to include them.

169 The distinction between *setjmp()/longjmp()* and *sigsetjmp()/siglongjmp()* is only
 170 significant for programs which use the *sigaction()*, *sigprocmask()*, or *sigsuspend()*
 171 functions.

172 BSD systems provide functions named *_setjmp()* and *_longjmp()* which, together with
 173 *setjmp()/longjmp()*, save and restore signal masks. While many other systems provide
 174 versions of these functions that do not, the Working Group decided not to specify the
 175 relation of these functions to signal masks and to define a new set of functions instead.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

176 B.8.3.2 Specify Signal Handling

177 The *sigaction()* §3.3.4 was introduced in order to provide an interface for reliable signal
 178 handling (see *Signals* §B.3.3). The *signal()* function is included in this document
 179 because *signal()* is defined in the *ANSI/X3.159-198x Programming Language C*
 180 *Standard*. However, it is recommended that POSIX applications use only the *sigaction()*
 181 interface, due to the potential unreliability and lack of consistency among existing
 182 *signal()* implementations. Portable library routines often need to install a signal catching
 183 function and then restore the signal to its original state. The function *sigaction()* should
 184 always work correctly for this purpose, regardless of what the rest of the program does.
 185 The *signal()* function may not work correctly if other parts of the program use
 186 *sigaction()*.

187 It is the intention of the Working Group that *signal()* be implementable as a library
 188 routine using *sigaction()*.

189 B.9 System Databases

190 At one time, this chapter was entitled *Passwords*, but this title was changed as all
 191 references to a “password file” were changed to refer to a “user database.”

192 B.9.1 System Databases

193 There are no references in the standard to a *passwd* file §B.2.3 or a *group* file §B.2.3 and
 194 there is no requirement that the *group* or *passwd* databases be kept in ASCII files. Many
 195 large timesharing systems use *passwd* databases that are hashed for speed. Certain
 196 security classifications prohibit certain information in the *passwd* database from being
 197 publicly readable.

198 The encoded password fields were deleted from both the *passwd* and *group* databases in
 199 order to meet the requirements of the US Government NBS Password FIPS (and FIPS
 200 concerns in general).

201 The term “encoded” is used instead of “encrypted” in order to avoid the
 202 implementation connotations (such as reversability, or use of a particular algorithm) of
 203 the latter term.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

204	B.9.2 Database Access	
205	B.9.2.1 Group Database Access	
206	B.9.2.2 User Database Access	C
207	B.10 Data Interchange Format	
208	B.10.1 Archive/Interchange File Format	
209	There are three areas of interest associated with file interchange:	B
210	Media	
211	There are other existing standards that define the media used for data interchange.	B
212	User Interface	B
213	This rightfully should be in the IEEE Std 1003.2 standard.	B
214	Format of the Data	B
215	None of the P1003 Working Groups address topics that match this area. The Working Group feels that this area is closest to the types of things that should be in the IEEE Std 1003.1 document, as the level of that document most closely matches the level of data required.	B
216		B
217		B
218		B
219		B
220	There appear to be two programs in wide use today, <code>tar</code> and <code>cpio</code> . There are large camps of supporters for each program. Four options were considered for the standard:	B
221		B
222	1. Make both formats optional. This was considered unacceptable because it does not allow any portable method for data interchange.	B
223		B
224	2. Require one format.	B
225	3. Require one format with the other optional.	B
226	4. Require both formats.	B
227	This issue is not yet resolved. In the September 1987 meeting, the <code>cpio</code> format was approved for inclusion in the standard as the data interchange format. The Extended <code>tar</code> Format was placed into Appendix D to solicit Balloting Group opinions on this issue.	C
228		C
229		C
230	There are a number of concerns about defining extensions that are known to be required by existing implementations. Failure to specify a consistent method to implement these extensions will severely limit portability of the program and, more importantly, will create severe confusion if these extensions are later standardized.	B
231		B
232		B
233		B
234	Two of these extensions that the Working Group felt should be documented are symbolic links, that were defined by 4.2BSD and 4.3BSD systems, and high performance (or contiguous) files, that exist in a number of implementations and are now being considered for the 1003.4 standard.	B
235		B
236		B
237		B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

238 By defining these extensions, implementors are able to recognize these features and take B
239 appropriate implementation defined actions for these files. For example, a high B
240 performance file could be converted to a regular file if the system didn't support high B
241 performance files; symbolic links might be replaced by normal hard links. B

242 The Working Group has held to the policy of not defining user interfaces to utilities by B
243 avoiding any description of a `tar` or `cpio` command. The behavior of the former B
244 command was described in some detail in previous drafts. B

245 The possibilities for transportable media include, but are not limited to,

- 246 1. 1/2 inch magnetic tape, 9 track, 1600 BPI
- 247 2. 1/2 inch magnetic tape, 9 track, 6250 BPI
- 248 3. Qic-11, 1/4 inch streamer tape
- 249 4. Qic-24, 1/4 inch streamer tape
- 250 5. 5.25 inch floppies, 8 512-byte sectors/track, 96 TPI
- 251 6. 5.25 inch floppies, 8 512-byte sectors/track, 48 TPI
- 252 7. IBM 3480 cartridges.

253 Specification of such media was considered part of the scope of the Trial Use Standard,
254 but has been excluded from the Full Use Standard.

255 The utilities are not restricted to work only with *transportable* media: existing related
256 utilities are often used to transport data from one place to another in the file hierarchy.

257 The format is included to provide an implementation independent way to move files from 9
258 one system to another and also to provide a way for a user to save data on a transportable 9
259 medium to be restored at a later date. Unfortunately, these two goals can contradict each 9
260 other as system security problems are easy to find in tape systems if they are not 9
261 protected. Thus the strict requirements about how the mechanism to copy files shall react c
262 when operated by both privileged and nonprivileged users. The general concept is that a
263 privileged (using the ISUID bit in the file's mode with the owner UID of the file set to
264 super-user) version of the utility can be used as a save/restore scheme, but a
265 nonprivileged version is used to interpret media from a different system without
266 compromising system security.

267 Regardless of the archive format used, guidelines should be observed when writing tapes c
268 to be read on other systems. Assuming the target system is POSIX compliant, archives c
269 created should use only use definitions found in POSIX (e.g., file types, minimum values c
270 as found in Chapter 2) and should only use relative pathnames (i.e., no leading /). c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

271 B.10.1.1 `cpio` Archive Format

272 The model for this format is the existing System V `cpio -c` data interchange format.
 273 This models documents the portable version of `cpio` format and *not* the binary version.
 274 It has the flexibility to transfer data of any type described within the POSIX standard, yet
 275 is extensible to transfer data types specific to extensions beyond POSIX (e.g., symbolic
 276 links or contiguous files). Because it describes existing practice, there is no question of
 277 maintaining upward compatibility.

278 This section does not standardize behavior for the utility when the file type is not
 279 understood or supported. It is useful for the utility to report to the user whatever action is
 280 taken in this case, though the standard neither requires nor recommends this.

281 B.10.1.1.1 Header

282 There has been some concern that the size of the `c_ino` field of the header is too small to
 283 handle those systems which have very large i-node numbers. However, the `c_ino` field in
 284 the header is used strictly as a hard link resolution mechanism for archives. It is not
 285 necessarily the same value as the i-node number of the file in the location that file is
 286 extracted from.

287 B.10.1.1.2 File Name

288 For most current implementations of the `cpio` utility, `{PATH_MAX}` bytes can be used
 289 to describe the pathname without the addition of any other header fields (the null byte
 290 would be included in this count). `{PATH_MAX}` is the minimum value for pathname
 291 size, documented as 256 bytes in Chapter 2 of the standard. However, an
 292 implementation may use `c_namesize` to determine the exact length of the pathname.
 293 With the current description of the `cpio` header, this pathname size can be as large as a
 294 number which is described in six octal bytes.

295 B.10.1.1.3 File Data

296 B.10.1.1.4 Special Entries

297 These are provided to maintain backward compatibility.

298 B.10.1.1.5 `cpio` Values

299 Three values are documented under the `c_mode` field values to provide for extensibility
 300 for known file types:

301 110000 Suggested symbolic name—`ISCTG`; reserved for contiguous files.
 302 The implementation may treat the rest of the information for this
 303 archive like a regular file. If this file type is undefined, the
 304 implementation may create the file as a regular file.

305 120000 Suggested symbolic name—`ISLNK`; reserved for files with
 306 symbolic links. The implementation may store the link name
 307 within the data portion of the file. If this type is undefined, the
 308 implementation may not know how to link this file or be able to
 309 understand the data section. The implementation may decide to

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 310 ignore this file type and output a warning message. c
- 311 140000 Suggested symbolic name—ISSOCK; reserved for sockets. If this c
 312 type is undefined on the target system, the implementation may c
 313 decide to ignore this file type and output a warning message. c
- 314 This provides for extensibility of the `cpio` format while allowing for the ability to read c
 315 old archives. Files of an unknown type may be read as “regular files” on some c
 316 implementations. c
- 317 **B.10.1.2 Multiple Volumes**
- 318 Multi-volume archives have been introduced in a manner that has become a *de facto* 9
 319 standard in many implementations. Though it is not required by POSIX classical 9
 320 implementations of the format-reading and -creating utility, upon reading logical end-of- 9
 321 file, check to see if an error channel is open to a controlling terminal. The utility then 9
 322 produces a message requesting a new medium to be made available. The utility waits for 9
 323 new medium to be made available by attempting to read a message to restart from the 9
 324 controlling terminal. In all cases, the communication with the controlling terminal is in 9
 325 an implementation defined manner. 9
- 326 The section **Multiple Volumes §10.1.2** is intended to handle the issue of multiple c
 327 volume archives. Since the end-of-medium and transition between media are not c
 328 properly part of this standard, the transition is described in terms of files. c
- 329 The intent is that files will be read serially until the end-of-archive indication is c
 330 encountered, and that file or media change will be handled by the utilities in an c
 331 implementation defined manner. c
- 332 Note that there was an issue with the representation of this on magnetic tape, and the c
 333 standard is intended to be interpreted such that each byte of the format is represented on c
 334 the media exactly once. In some current implementations, it is not deterministic whether c
 335 encountering the end-of-medium reflector foil on magnetic tape during a write will yield c
 336 an error during a subsequent `read()` of that record, or if that record is actually recorded. c
 337 on the tape. It is also possible that `read()` will encounter the end-of-medium when end- c
 338 of-medium was not encountered when the data was written. This has to do with c
 339 conditions where the end of [magnetic] record is in such a position that the reflector foil c
 340 is on the verge of being detected by the sensor and is detected during one operation and c
 341 not on a later one, or vice-versa. c
- 342 An implementation of the format-creating utility must assure when it writes a record that c
 343 the data appears on the tape exactly once. This implies that the program and the tape c
 344 driver work in concert. An implementation of the format-reading utility must assure that c
 345 an error in a boundary condition described above will not cause loss of data. c
- 346 The general consensus was that the following would be considered as correct operation c
 347 of a tape driver when end-of-medium is detected: c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

348 During writing, either:

349 1. The record where the relector spot was deleted is backspaced over
350 by the driver so that the trailing tape mark that will be written on
351 *close()* will overwrite.

352 Writing the tape mark should not yield an end-of-medium
353 condition.

354 2. Or, the condition is reported as an error on the *write()* following the
355 one where the end-of-medium is detected (the one where the end-
356 of-medium is actually detected completing successfully). No data
357 will be actually transferred on the *write()* reporting the error. The
358 subsequent *close()* would *write()* a tape mark following the last
359 record actually written.

360 Writing the tape mark, and writing any subsequent records, should
361 not yield any end-of-medium conditions.

362 (The latter behavior permits the implementation of ANSI standard labels
363 because several records (the trailer records) can be written after the end-
364 of-medium indications. It also permits dealing with, for example, COBOL
365 "ON" statements.)

366 During reading:

367 The end-of-medium indicator is simply ignored, presuming that a tape
368 mark (end-of-file) will be recorded on the magnetic medium, and the
369 reflector foil was advisory only to the *write()*.

370 Systems where these conditions are not met by the tape driver should assure that the
371 format-creating and -reading utilities assure proper representation and interpretations of
372 the files on the media, in a way consistent with the above recommendations.

373 The typical failures on systems that do not meet the above conditions are either:

374 1. To leave the record written when the end-of-medium is encountered on the
375 tape, but to report that it was not written. The format-creating utility would
376 then rewrite it, and then the format-reading utility could see the record
377 twice if the end-of-medium is not sensed during the read operations.

378 2. Or, the *write()* occurs uneventfully, but the *read()* senses the error and
379 does not actually see the data, causing a record to be omitted.

380 Nothing in this standard requires that end-of-medium be determined by anything on the
381 medium itself (for example, a predetermined maximum size would be an acceptable
382 solution for the format creating utility). The format-reading utility must be able to
383 *read()* tapes written by machines that do use the whole medium, however.

384 On media where end-of-medium and end-of-file are reliably coincident, such as disks, c
 385 end-of-medium and end-of-file can be treated as synonyms. c

386 Note that partial physical records (corresponding to a single *write()*) can be written on c
 387 some media, but that only full physical records will actually be written to magnetic tape, c
 388 given the way the tape operates. c

389 **B.10.1.3 Extended tar Format** c

390 This section was originally in the body of the Trial Use Standard but was moved to c
 391 Appendix D for the Full Use Ballot. c

392 The original model for this facility is the 4.3BSD or Version 7 *tar* program and format, c
 393 but the format given here is an extension of the traditional *tar* format. The name c
 394 USTAR was adopted to reflect this.

395 This description reflects numerous enhancements over previous versions. The goal of 9
 396 these changes was not only to provide the functional enhancements desired, but to retain 9
 397 compatibility between new and old versions. This compatibility has been retained. 9
 398 Archives written using the old archive format are compatible with the new format. 9
 399 Archives written using this new format may be read by applications designed to use the 9
 400 old format as long as the functional enhancements provided here are not used. This 9
 401 means the user is limited to archiving only regular type files and nonsymbolic links to 9
 402 such files. 9

403 If a utility reads an archive that contains file types that the utility either does not
 404 understand or does not support (such as symbolic links or contiguous files), it is useful
 405 for the utility to report whatever action it takes to the user, though the standard neither
 406 requires nor recommends this.

407 Implementors should be aware that the previous file format did not include a mechanism 9
 408 to archive directory type files. For this reason, the convention of using a file name B
 409 ending with slash was adopted to specify a directory on the archive. 9

410 Note that, NAMSIZ plus PFXSIZ have been set to meet the minimum requirements for 9
 411 {PATH_MAX}. If a pathname is less than NAMSIZ-1 characters and therefore fits within 9
 412 the *name* field, it is recommended that the pathname be stored there without the use of 9
 413 the *prefix* field. Although the value of NAMSIZ is known to be less than {PATH_MAX}, 9
 414 the value was not changed in this version of the archive file format to retain backward 9
 415 compatibility and instead the *prefix* was introduced. Also because of the earlier version 9
 416 of the format, there is no way to remove the limitation on the *linkname* field being set to 9
 417 NAMSIZ. 9

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

418 **B.11 Bibliographic Notes**

419 There are far more related papers and books than are mentioned here, and some of them
420 may be as good or better.

421 **B.11.1 Related Standards**

422 The standard assumes that any terms not defined in Chapter 2 are defined in the *IEEE*
423 *Standard Dictionary of Electrical and Electronics Terms*, IEEE Std 100-1977. B
B

424 The *1984 /usr/group Standard* may be ordered from

425 /usr/group Standards Committee A
426 4655 Old Ironsides Drive, Suite 200 A
427 Santa Clara, California 95054 A
428 (408)986-8840 A

429 The basic historical reference on the C language is

430 • Kernighan, Brian W. and Ritchie, Dennis M., *The C Programming Language*,
431 Prentice-Hall, Englewood Cliffs, New Jersey (1978).

432 The *ANSI/X3.159-198x Programming Language C Standard* may be obtained from

433 Global Press A
434 2625 Hickory St. A
435 P.O. Box 2504 A
436 Santa Anna, CA 92707-3783 A
437 U.S.A. A

438 800-854-7179 A
439 +1-714-540-9870 (from outside the U.S., ask for extension 245.) A
440 TELEX 692 373 A

441 The *X/OPEN Portability Guide* is published by

442 Elsevier Science Publishers B.V. A
443 Book Order Department A
444 P.O. Box 1991 A
445 1000 BZ Amsterdam A
446 The Netherlands A

447 and is distributed in the United States and Canada by

448 Elsevier Science Publishing Company, Inc. A
449 52 Vanderbilt Avenue A
450 New York, NY 10017 A
451 U.S.A. A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

452 There are five volumes, of which Volume 2 is the most relevant to the present standard.

453 B.11.2 Historical Implementations

454 A principal ancestor of all the historical implementations is the Multics System

- 455 • Organick, Elliot I., *The Multics System: An Examination of Its Structure*, The MIT
456 Press, Cambridge, MA (1975).

457 The most basic and influential paper on historical implementations is

- 458 • Ritchie, D. M. and Thompson, K., "The UNIX Time-Sharing System," *Bell*
459 *System Technical Journal* 57(6 Part 2) pp. 1905-1929 American Telephone and
460 Telegraph Company, (July-August 1978). This is a revised version and describes
461 Version 7.

- 462 • Ritchie, D. M. and Thompson, K., "The UNIX Time-Sharing System," *Commun.*
463 *ACM* 7(7) pp. 365-375 Association for Computing Machinery, (July 1974). This is
464 the original paper, which describes Version 6.

465 The Version 7 manual is

- 466 • AT&T, *UNIX Time Sharing System: UNIX Programmer's Manual, Seventh Edition*,
467 Bell Telephone Laboratories, Inc., Murray Hill, New Jersey (January, 1979).

468 Dennis Ritchie has also done several papers on the history and evolution of the system

- 469 • Ritchie, Dennis, "The Evolution of the UNIX Time-sharing System," *AT&T Bell*
470 *Laboratories Technical Journal* 63(8) pp. 1577-1593 American Telephone and
471 Telegraph Company, (October 1984).

- 472 • Ritchie, Dennis M., "Reflections on Software Research," *Commun. ACM* ?(?) p. ?
473 Association for Computing Machinery, (1984). ACM Turing Award Lecture

- 474 • Ritchie, Dennis M., "Unix: A Dialectic," *USENIX Association Conference*
475 *Proceedings*, pp. 29-34 USENIX Association, P.O. Box 2299, Berkeley, CA
476 94710, (21-23 January 1987).

477 Important collections of papers on the system may be found in

- 478 • BSTJ, "UNIX Time-Sharing System," *Bell System Technical Journal* 57(6 Part
479 2) American Telephone and Telegraph Company, (July-August 1978).

- 480 • BLTJ, "The UNIX System," *AT&T Bell Laboratories Technical Journal*
481 63(8) American Telephone and Telegraph Company, (October 1984).

482 The System III manual is

- 483 • AT&T, *UNIX System III Programmer's Manual*, Western Electric Company, Inc.,
484 Greensboro, N.C. (October, 1981).

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

485 The *SVID*

486 • AT&T, *System V Interface Definition, Issue 2*, AT&T (1986).

487 may be ordered from

488 AT&T Customer Information Center
 489 Attn: Customer Service Representative
 490 P.O. Box 19901
 491 Indianapolis, IN 46219
 492 U.S.A.

493 800-432-6600 (Inside U.S.A.)
 494 800-255-1242 (Inside Canada)
 495 317-352-8557 (Outside U.S.A. and Canada)

496 using the following Select Codes:

497 320-011 Volume I
 498 320-012 Volume II
 499 320-013 Volume III
 500 307-131 all three volumes

501 The implementation of System V is described in

502 • Bach, Maurice J., *The Design of the UNIX Operating System*, Prentice-Hall,
 503 Englewood Cliffs, New Jersey (1986).

504 The 4.3BSD manual

505 • UCB-CSRГ., *4.3 Berkeley Software Distribution, Virtual VAX-11 Version*, The
 506 Regents of the University of California, Berkeley, California (April 1986).

507 is printed by the USENIX Association, and their members may order from them:

508 USENIX Association
 509 P.O. Box 2299
 510 Berkeley, CA 94710
 511 415-528-8649

512 The implementation of the kernel of 4.3BSD is described in

513 • Quarterman, John S., Silberschatz, Abraham, and Peterson, James L., "4.2BSD and
 514 4.3BSD as Examples of the UNIX System," *ACM Computing Surveys* 17(4) pp.
 515 379-418 Association for Computing Machinery, (December 1985).

516 • Leffler, Samuel J., McKusick, Marshall Kirk, Karels, Michael J., Quarterman, John
 517 S., and Stettner, Armando, *The Design and Implementation of the 4.3BSD UNIX*
 518 *Operating System*, Addison-Wesley, Reading, Massachusetts (1988).

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

519 **B.11.3 Historical Application Programming Tutorials**

520 A useful tutorial on programming in the C language is

- 521 • Harbison, Samuel P. and Steele, Guy L.,
- C: A Reference Manual*
- , Prentice-Hall,
-
- 522 Englewood Cliffs, New Jersey (1987).

523 A highly regarded book, though not one for beginners, is

- 524 • Kernighan, Brian W. and Pike, Rob,
- The UNIX Programming Environment*
- ,
-
- 525 Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1984).

526 One more oriented towards Berkeley systems is

- 527 • McGilton, Henry and Morgan, Rachel,
- Introducing the UNIX System*
- , McGraw-Hill
-
- 528 (BYTE Books), New York (1983).

529 and a more recent one is

- 530 • Rochkind, Marc J.,
- Advanced UNIX Programming*
- , Prentice-Hall, Englewood
-
- 531 Cliffs, New Jersey (1985).

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

C. Comparison to *System V Interface Definition*

- 1 The *System V Interface Definition (SVID)* defines the external characteristics (externally c
2 visible interfaces and behavior) common to all System V environments. When it was c
3 first published in 1984, it differed in small ways with the *1984 lusr/group Standard*, and
4 those differences were listed in Issue 1 of the *SVID*. This appendix lists the differences
5 between Issue 2 of the *SVID* (Volumes 1-3) and the IEEE Std 1003.1. Unless otherwise c
6 noted, all differences are compared to the BASE definition of the *SVID*. Overall
7 differences are described first and then differences in specific functions are described.
8 All known differences in defined functionality are listed although some may be of minor
9 importance.
- 10 In most cases, on a specific point of difference, both IEEE Std 1003.1 and *SVID*
11 definitions are presented. In other cases, particularly when one document includes a
12 point that the other does not, only the statement from that document is characterized.
- 13 c
- 14 Numbers in parentheses below, such as (2.3) or (3.2.2.2), refer to sections in IEEE Std
15 1003.1.

16 C.1 Overall Contents

17 C.1.1 Operating System Primitives

18 Functions included only in

19 1003.1: *mkfifo()*, *getgroups()*, *rename()*, *pathconf()*, *fpathconf()*, *sysconf()*. c

20 c

21 *mkfifo()*, *pathconf()*, and *sysconf()* are new functions. c

22 In System V, FIFO files are made with the *mknod()* function. c

23 The optional *getgroups()* function is not included in the *SVID*, c

24 The *rename()* function is not included in the *SVID*. c

25 *SVID*: *ioctl()*, *mknod()*, *mount()*, *umount()*, *pclose()*, *popen()*, *stime()*, *sync()*,
26 *ulimit()*, *ustat()*.

27 The *SVID* defines these ten additional functions and requires them to be
28 supported by any System V environment.

29 8

30 C.1.2 Library Routines

31 Functions described only in:

32 1003.1: Eleven routines are included in 1003.1 that are not found in the Base System c
33 definition in the *SVID*, but are found in the Software Development Extension.
34 These include five routines that access the group database (*/etc/group* in
35 *SVID*): *endgrent()*, *getgrent()*, *setgrent()*, *getgrid()*, *getgrnam()*; five
36 routines that access the passwd database (*/etc/passwd* in *SVID*): *endpwent()*,
37 *getpwent()*, *setpwent()*, *getpwnam()*, *getpwuid()*; one routines to return user
38 login names, *getlogin()*. One routine is included in 1003.1 that is not in the c
39 Software Development Extension in the *SVID*: *cuserid()*. c

40 *SVID*: The *SVID* defines approximately 150 additional routines many of which are
41 covered in the *ANSI/X3.159-198x Programming Language C Standard*. and 8
42 are included in 1003.1 by reference (8.1). Any differences between the *SVID*
43 definitions and the *ANSI/X3.159-198x Programming Language C Standard* 8
44 definitions are not covered in this appendix. These include math routines,
45 memory allocation, non-local jumps, data conversion and encoding, *stdio*
46 routines, string and character handling, sorting, regular expression matching,
47 search routines and some others.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

48 C.1.3 Special Files

49	<i>SVID</i> :	Three special device files are required by the <i>SVID</i> ,	
50		/dev/console	system console interface
51		/dev/null	the null file
52		/dev/tty	controlling terminal interface.

A
A
A

53 C.1.4 Minimal Directory Tree Structure

54	<i>SVID</i> :	Specifies a minimal directory tree structure comprising /bin, /dev, /etc,
55		/tmp, /usr/bin, and /usr/tmp.

56 C.1.5 Multiple Groups

57	1003.1:	Defines supplemental groups as an optional feature ({NGROUPS_MAX} may
58		be zero). This feature affects several components of the standard.

C

59 8

60 C.1.6 Job Control

8

61	1003.1:	Defines job control as an optional feature. None of the functions detailed	8
62		here are included unless the Job Control Option is present. This feature	A
63		affects several components of the standard: four functions (<i>jcsetpgrp()</i> ,	B
64		<i>tcgetpgrp()</i> , <i>tcsetpgrp()</i> , and <i>wait2()</i>) and a header file (<wait.h>) have	8
65		been added to the standard. In addition, the <i>signal()</i> definition was affected	A
66		and other signals were added.	A

67	<i>SVID</i> :	Does not include the Job Control option.
----	---------------	--

68 C.1.7 Enhanced Signals

69	1003.1:	(3.3.3) Extends the signal handling functions to include a set of functions	
70		that manage sets of signals. The functions <i>siginitset()</i> , <i>sigfillset()</i> ,	B
71		<i>sigaddset()</i> , <i>sigdelset()</i> , <i>sigismember()</i> , <i>sigaction()</i> , <i>sigprocmask()</i> ,	B
72		<i>sigpending()</i> , <i>sigsuspend()</i> were added to the standard. The structure	8
73		definition <i>sigaction</i> was added to the header file <signal.h>.	8

74	1003.1:	Specifies that the signal mask is conditionally saved and restored by the	B
75		<i>sigsetjmp()</i> and <i>siglongjmp()</i> functions.	B

76	<i>SVID</i> :	Volume 3 added functions to support an extended form of signal handling.	8
77		The functions <i>sigset</i> , <i>sighold</i> , <i>sigrelse</i> , and <i>sigignore</i> were added. All	8
78		functions takes a single signal number of type <i>int</i> . The <i>sigset</i> function takes	8

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 79 an additional parameter which is one of four values: SIG_DFL, SIG_IGN, 8
80 SIG_HOLD, or an *address* of a signal-catching function. C
- 81 **C.1.8 Configurable System Variables** C
- 82 1003.1: Three new functions, *fpathconf()*, *pathconf()*, and *sysconf()*, were added to C
83 the system configuration variables. C
- 84 **C.1.9 Terminal I/O**
- 85 The comparison described here is between *termios* from 1003.1 and *termio* from *SVID*.
- 86 1003.1: (7.1) Specifies a set of functions to manipulate a terminal.
- 87 *SVID*: Specifies a set of *ioctl* commands to manipulate a terminal.
- 88 **C.2 Specific Differences**
- 89 **C.2.1 Error Numbers**
- 90 1003.1: (2.5) Includes the additional errors
- 91 ENAMETOOLONG filename too long A
92 ENOTEMPTY directory not empty C
- 93 *SVID*: Includes the additional error C
- 94 ENOTBLK block device required C
95 ETXTBSY program text file busy 8
- 96 *SVID*: Volume 3 of the *SVID* specifies as a future direction, that in the case of a 8
97 path-name argument exceeding {PATH_MAX}, the error returned would C
98 change to follow the direction of the 1003.1 standard. Volume 3 currently
99 specifies ENOENT as the error returned.

100 C.2.2 General Terms

- 101 8
- 102 1003.1: (2.4) *pathname searches*—As a special case, in the root directory, “dot- 8
103 dot” may refer back to root directory itself. 8
- 104 *SVID*: *directory*—The root directory, which is the top-most node of the hierarchy, 8
105 has itself as its parent directory. 8

106 C.2.3 Data Types

- 107 1003.1: The defined type *time_t* is time measured in seconds and *clock_t* is time 8
108 measured in {CLK_TCK}ths of a second. (2.6)
- 109 *SVID*: The defined type *time_t* is time measured in either {CLK_TCK}ths of a 8
110 second (*times()*) or in seconds (*stat()*). The type *clock_t* is not defined in 8
111 *SVID*. A
- 112 1003.1: The defined type *uid_t* is used to represent user and group IDs. As a result, c
113 differences in synopses exist in the following functions: *getuid()*, *geteuid()*, c
114 *getgid()*, *getegid()*, *setuid()*, *setgid()*, <sys/stat.h>, and *chown()*.
- 115 1003.1: The defined type *mode_t* is used to represent file modes. As a result, c
116 differences in synopses exist in the following functions: *creat()*, *umask()*, c
117 *mkdir()*, *open()*, <sys/stat.h>, and *chmod()*. B

118 C.2.4 Environment Variables

- 119 1003.1: (2.7) Defines additional variables that may be defined: PS1, PS2, IFS, MAIL, c
120 SHELL, LOGNAME, LC_CTYPE, LC_COLLATE, LC_TIME, LC_TIME, and c
121 LC_NUMERIC.

122 C.2.5 fork()

- 123 1003.1: (3.1.1.2) Lists attributes *not* inherited by the child process and specifies that 8
124 all other attributes defined by the standard shall be inherited. 8
125 Implementations may add characteristics that are or are not inherited. 8
- 126 *SVID*: Lists attributes that must be inherited as well as those not inherited by the
127 child process. A

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

128 C.2.6 `exec()`

129 *SVID*: When a C program is executed, it is called as follows

```
130         main (argc, argv, envp)
131         int argc;
132         char **argv, **envp;
```

133 In 1003.1, (3.1.2.2) the third argument, is not specified. B

134 *SVID*: The effective user ID and group ID of the new process are saved for use by `setuid()`. In 1003.1, (3.1.2.2) this is optional.

136 *SVID*: Specifies that the new process additionally inherits the terminal group id and file-size limit of the calling process. 9

137 9

138 8

139 8

140 C.2.7 `wait()`

141 1003.1: (3.2.1.2) If the child process terminated due to a signal that was not caught,
142 the low order 6 bits of *status* will contain the signal number.

143 *SVID*: If the child process terminated due to a signal that was not caught, the low
144 order 7 bits of *status* will contain the signal number.

145 1003.1: Additionally allows `wait()` to return due to an implementation-defined
146 change in the status of a child process. A

147 C.2.8 `_exit()`

148 1003.1: (3.2.2.2) If the calling process is the process group leader, `SIGHUP` may be
149 sent to each process with a process group ID equal to the calling process.

150 *SVID*: If the calling process is a process group leader and is associated with a
151 controlling terminal, `SIGHUP` is sent to each process with a process group ID
152 equal to that of the calling process. 8

153 1003.1: If a child process is stopped under job control, it will be sent both `SIGHUP`
154 and `SIGCONT`. 8

155 C.2.9 <signal.h>

- 156 1003.1: (3.3.1.2) The additional signal SIGSEGV is defined. B
- 157 SVID: The signal SIGSEGV is not on the list of signals that applications should B
 158 know about and the SVID warns that its meaning is implementation- B
 159 dependent. B
- 160 SVID: The additional signal SIGSYS, bad argument to system call, is defined. This B
 161 signal is not in 1003.1. B
- 162 SVID: The signal SIGABRT defined in 1003.1 is indicated in SVID Volume 3. 8
- 163 1003.1 The signals SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, and SIGCONT are 8
 164 optional based on the presence of the Job Control Option. 8

165 C.2.10 kill()

- 166 8
- 167 SVID: Specifies that an error is returned if the arguments *sig* is SIGKILL and *pid* is B
 168 a special system process. 8
- 169 1003.1 (3.3.2.2) Specifies that if the signal is being sent to all processes, the sender 8
 170 *may* be excluded. 1003.1 also specifies that if both {_POSIX_KILL_SAVED} c
 171 and {_POSIX_SAVED_IDS} are defined, the saved set-user-ID of the c
 172 receiving process shall be checked in place of its effective user ID. c

173 C.2.11 signal()

- 174 1003.1: (3.3.8.2) A call to *signal()* shall cancel a pending signal if the *func* c
 175 parameter is SIG_IGN, and may cancel pending signals, except for a pending
 176 SIGKILL signal.
- 177 SVID: A call to *signal()* cancels a pending signal of *type sig* except for a pending
 178 SIGKILL signal. (Note that only a pending signal of the same type for which
 179 signal was just called is affected.)

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

180 C.2.12 times()

181 1003.1: (4.5.2.2) Specifies the members of the *tms* structure as type `clock_t`.182 SVID: Specifies the members of the *tms* structure as type `time_t`. A

183 C.2.13 open()

184 1003.1: (5.3.1.2) When a file is created with the `O_CREAT` flag, 1003.1 specifies that c
185 the file's group ID shall be set to either the process's effective group ID *or* to
186 the group ID of the directory in which the file is being created.187 SVID: Specifies that when a file is created with the `O_CREAT` flag, the file's group
188 ID is set to the process's effective group ID.189 1003.1: Specifies the flag `O_NONBLOCK`.190 SVID: Specifies the flag `O_NDELAY`.

191 SVID: Specifies two additional error conditions.

192 ENXIO The named file is a character special or block special file and
193 the device associated with the special file does not exist.194 ETXTBSY The file is a pure procedure (shared text) file that is being
195 executed and *oflag* is write or read/write.

196 C.2.14 unlink()

197 SVID: Specifies the additional error condition

198 ETXTBSY The entry to be unlinked is the last link to a pure procedure
199 file that is being executed.

200 C.2.15 rmdir()

201 1003.1: (5.5.2.1) Specifies that an implementation can return either `EEXIST` or B
202 `ENOTEMPTY` if the directory being removed contains files. B203 SVID: Specifies that an implementation shall return `EEXIST` if the directory being B
204 removed contains files. B

205	C.2.16	<sys/stat.h>		B
206	1003.1:	Recommends the S_ISUID and the S_ISGID bits be cleared on every <i>write</i> .		C
207				C
208	C.2.17	access()		C
209	1003.1:	Specifies the optional error condition		C
210		EINVAL Invalid value for <i>amode</i> .		
211	SVID:	Specifies the additional error condition		
212		ETXTBSY Write access requested for a pure procedure file that is being		
213		executed.		
214				9
215				C
216	C.2.18	chown()		
217	1003.1:	(5.6.5.4) Specifies the optional error condition		C
218		EINVAL The owner or group ID supplied is outside the range of 0 to		
219		{UID_MAX}, inclusive.		
220	C.2.19	utime()		
221	1003.1:	(5.6.6.2) Specifies the inclusion of <utime.h> which defines the <i>utimbuf</i>		
222		structure.		
223	SVID:	The <i>utimbuf</i> structure must be defined by the user.		
224	C.2.20	close()		
225	1003.1:	(6.3.1.1) Specifies the additional error condition		
226		EINTR The <i>close</i> function was terminated prematurely by a signal.		

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

227 C.2.21 read()

228

229 1003.1: (6.4.1.4) Specifies that an error will be returned and *errno* set to EAGAIN if
 230 no-delay (O_NONBLOCK) mode is in effect and the process would be
 231 delayed in reading.

232 SVID: SVID Volume 3 specifies read will return 0 in the no-delay (O_NDELAY)
 233 case. The change to return EAGAIN is listed as a future direction.

234

235 SVID: Specifies the additional errors

236 EIO A physical I/O error has occurred

237 ENXIO The device associated with the file descriptor is a block-
 238 special or character-special file and the value of the file
 239 pointer is out of range.

240 C.2.22 write()

241

242 1003.1: (6.4.2.4) Specifies the additional error condition:

243 EAGAIN O_NONBLOCK is set and the process would be delayed in the
 244 *write()* operation

245 SVID: Specifies the additional errors

246 EIO A physical I/O error has occurred

247 ENXIO The device associated with the file descriptor is a block-
 248 special or character-special file and the value of the file
 249 pointer is out of range.

250 SVID: Specifies that in the O_NDELAY case, if the write request doesn't transfer
 251 data, 0 is returned.

252 C.2.23 <fcntl.h>

253 1003.1: (6.5.1.2) Specifies the symbolic name of the no-delay flag to be
 254 O_NONBLOCK.

255 SVID: Specifies the symbolic name of the no-delay flag to be O_NDELAY.

256 C.2.24 fcntl()

257 1003.1: (6.5.2). Specifies the additional error condition c

258 EINTR The *fcntl* function was terminated prematurely by a signal.

259 C.2.25 lseek()

260 1003.1: (6.5.3.1) Specifies the function and its argument *offset* to be of type *off_t*.

261 SVID: Specifies the function and its argument *offset* to be of type *long*. c

262 1003.1: Specifies the additional error condition

263 EINVAL The resulting file pointer would be illegal.

264 B

265 C.2.26 Terminal I/O c

266 1003.1: Specifies the terminal control structure *termios*. c

267 SVID: Specifies the terminal control structure *termio*. c

268 1003.1: The Job Control Option is described. This includes changing the process c
 269 group associated with the terminal, generating signals, SIGTTIN and c
 270 SIGTTOU, for reads and writes from processes outside of the distinguished c
 271 process group, generating a signal, SIGTSTP, upon receipt of a special c
 272 character, SUSP, and a control flag, TOSTOP. c

273 SVID: Volume 3 does not include the Job Control Option. c

274 A

275 c

276 1003.1: (7.1.2.2) Specifies the types of the mode elements as unsigned *long*.

277 SVID: Specifies the types as unsigned *short*. Specifies a line discipline element
 278 *c_line*.

279 9

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

280	<i>SVID:</i>	Specifies input mode flag <i>IUCLC</i> .	8
281			9
282	<i>SVID:</i>	Specifies output mode flags <i>OLCUC</i> , <i>ONLCR</i> , <i>OCRNL</i> , <i>ONOCR</i> ,	8
283		<i>ONLRET</i> , <i>OFILL</i> , <i>NLDLY</i> , <i>CRDLY</i> , <i>TABDLY</i> , <i>BSDLY</i> , <i>VTDLY</i> , and	8
284		<i>FFDLY</i> . Specifies delay values: <i>NL0</i> , <i>NL1</i> , <i>CR0</i> , <i>CR1</i> , <i>CR2</i> , <i>CR3</i> , <i>TAB0</i> ,	8
285		<i>TAB1</i> , <i>TAB2</i> , <i>TAB3</i> , <i>BS0</i> , <i>BS1</i> , <i>VT0</i> , <i>VT1</i> , <i>FF0</i> , and <i>FF1</i> .	8
286	1003.1:	(7.1.2.5) Specifies the macros <i>cf_getspeed()</i> , <i>cf_setospeed()</i> ,	9
287		<i>cf_getispeed()</i> , and <i>cf_setispeed()</i> that get and set the input and output	9
288		terminal speeds in a <i>termios</i> structure.	C
289			9
290	<i>SVID:</i>	Specifies the local mode flag <i>XCASE</i> .	C
291	1003.1:	(7.1.4) Specifies functions <i>tcgetattr()</i> and <i>tcsetattr()</i> .	
292	<i>SVID:</i>	Specifies commands and structures for use with <i>ioctl()</i> .	
293	1003.1:	(7.1.5) Specifies functions <i>tcsendbreak()</i> , <i>tcdrain()</i> , <i>tcflush()</i> , and <i>tcflow()</i> .	C
294		The send-break function has the option of sending zero-valued bits for a	
295		specified value. The flow function has control over input.	
296	<i>SVID:</i>	Specifies commands and structures for use with <i>ioctl()</i> .	
297			B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
Do not specify or claim conformance to this document.

D. Alternative Archive/Data Interchange Format

1 It has been proposed that the following section on the "Extended `tar` Format" be added c
2 to Chapter 10 as either an alternative to, or a replacement of, the "`cpio` Archive c
3 Format." Consult the cover letter for the ballot associated with this draft for an c
4 explanation of how to make your preferences known. Unless an explicit action is taken c
5 by the Balloting Group, this section will not appear in the approved Full Use Standard. c

6 D.1 Extended `tar` Format

7 An extended `tar` archive tape or file contains a series of blocks. Each block is a fixed c
8 size block of TBLOCK bytes (see below). Although this format may be thought of as c
9 being stored on 9-track industry standard 1/2-inch magnetic tape, other types of
10 transportable medium are not excluded. Each file archived is represented by a header
11 block that describes the file, followed by zero or more blocks that give the contents of the
12 file. At the end of the archive file are two blocks filled with binary zeros, interpreted as c
13 an end-of-archive indicator. c

14 The blocks may be grouped for physical I/O operations. Each group of n blocks (where n
15 is set by the application utility creating the archive file) may be written with a single
16 `write()` operation. On magnetic tape, the result of this write is a single tape record. The
17 last group of blocks is always at the full size, so blocks after the two zero blocks contain
18 undefined data.

19 The header block is structured as follows. All lengths and offsets are in decimal.

<u>Field Name</u>	<u>Byte Offset</u>	<u>Length (in bytes)</u>
<i>name</i>	0	100
<i>mode</i>	100	8
<i>uid</i>	108	8
<i>gid</i>	116	8
<i>size</i>	124	12
<i>mtime</i>	136	12
<i>chksum</i>	148	8
<i>typeflag</i>	156	1
<i>linkname</i>	157	100
<i>magic</i>	257	6
<i>version</i>	263	2
<i>uname</i>	265	32
<i>gname</i>	297	32
<i>devmajor</i>	329	8
<i>devminor</i>	337	8
<i>prefix</i>	345	155

39 Symbolic constants used in the header block are defined in the header <tar.h> as
40 follows:

```

41 #define TMAGIC "ustar" /* ustar and a null */
42 #define TMAGLEN 6
43 #define TVERSION "00" /* 00 and no null */
44 #define TVERSLEN 2

45 /* Values used in typeflag field */
46 #define REGTYPE '0' /* Regular file */
47 #define AREGTYPE '\0' /* Regular file */
48 #define LNKTYPE '1' /* Link */
49 #define SYMTYPE '2' /* Reserved */
50 #define CHRTYPE '3' /* Char. special */
51 #define BLKTYPE '4' /* Block special */
52 #define DIRTYPE '5' /* Directory */
53 #define FIFOTYPE '6' /* FIFO special */
54 #define CONTTYPE '7' /* Reserved */

55 /* Bits used in the mode field - values in octal */
56 #define TSUID 04000 /* Set UID on execution */
57 #define TSGID 02000 /* Set GID on execution */
58 #define TSVTX 01000 /* Reserved */
59 /* File permissions */
60 #define TUREAD 00400 /* read by owner */
61 #define TUWRITE 00200 /* write by owner */
62 #define TUEXEC 00100 /* execute/search by owner */

```

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

```

63 #define TGREAD      00040    /* read by group */           C
64 #define TGWRITE    00020    /* write by group */         C
65 #define TGEEXEC    00010    /* execute/search by group */ C
66 #define TOREAD     00004    /* read by other */         C
67 #define TOWRITE    00002    /* write by other */           C
68 #define TOEXEC     00001    /* execute/search by other */   C

```

69 All characters are represented in the American Standard Code for Information
70 Interchange, ASCII. For maximum portability between implementations, names should
71 be picked from characters represented by the portable filename character set §2.3 as
72 8-bit characters with zero parity. If an extended character set beyond the portable
73 character set is used, and the format-reading and format-creating utilities on the two
74 distinct systems use the same extended character set, the file name shall be preserved. C
75 However, the format-reading utility shall never create file names on the local system that
76 cannot be accessed via the functions calls described previously in this standard; see C
77 *open()* §5.3.1, *stat()* §5.6.2, *chdir()* §5.2.1, *fcntl()* §6.5.2, and *opendir()* §5.1.2. If a file C
78 name is found on the medium that would create an invalid file name, the implementation B
79 shall define if the data from the file is stored on the local file system and under what
80 name it is stored. A format-reading utility may choose to ignore these files as long as it C
81 produces an error stating that the file is being ignored. C

82 Each field within the header block is contiguous; that is, there is no padding used. Each C
83 character on the archive medium is stored contiguously.

84 The fields *magic*, *uname*, and *gname* are null-terminated character strings. The fields
85 *name*, *linkname*, and *prefix* are null-terminated character strings except when all
86 characters in the array contain non-null characters including the last character. All other
87 fields are leading zero-filled octal numbers in ASCII. Each numeric field (of width *w*)
88 contains *w*-2 digits, a space, and a null, except *size*, *mtime*, and *version*, that do not
89 contain the trailing null.

90 The *name* and the *prefix* fields produce the pathname of the file. The hierarchical
91 relationship of the file is retained by specifying the pathname as a path prefix, a slash B
92 character and filename as the suffix. If the *prefix* contains non-null characters, it is
93 concatenated in front of the *name* without modification or addition of new characters to
94 produce a new pathname. In this manner, pathnames of NAMSIZ plus PFXSIZ characters
95 can be supported. If a pathname does not fit in the space provided, the format-creating C
96 utility shall notify the user of the error, and no attempt shall be made by the format- C
97 creating utility to store any part of the file, header or data, on the medium. C

98 The *linkname* field, described below, does not use the *prefix* to produce a pathname. As
99 such, a *linkname* is limited to NAMSIZ minus one characters. If the name does not fit in C
100 the space provided, the format-creating utility shall notify the user of the error, and the C
101 utility shall not attempt to store the link on the medium. C

102 The *mode* field provides 9 bits specifying file permissions and 3 bits to specify the Set
103 UID, Set GID, and TSVTX modes. Values for these bits are defined above. When special C

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

104 permissions are required to create a file with a given mode, and the user restoring files
 105 from the archive does not hold such permissions, the mode bit(s) requiring those special
 106 permissions are ignored. Modes not supported by the implementation restoring the files
 107 from the archive are ignored.

108 The *uid* and *gid* fields are the user and group ID of the file's owner and group,
 109 respectively.

110 The *size* field is the size of the file in bytes. If the *type* flag field is set to specify a file to
 111 be of type LNKTYPE, the *size* field shall be specified as a zero (0).

112 The *mtime* field is the modification time of the file at the time it was archived. It is the
 113 ASCII representation of the octal value of the modification time obtained from the *stat()*
 114 function.

115 The *chksum* field is the ASCII representation of the octal value of the simple sum of all
 116 bytes in the header block. Each 8-bit byte in the header is treated as an unsigned value.
 117 These values are added to an unsigned integer, initialized to zero, the precision of which
 118 shall be no less than 17 bits. When calculating the checksum, the *chksum* field is treated
 119 as if it were all blanks.

120 The *typeflag* field specifies the type of file archived. If a particular implementation does
 121 not recognize or permit the specified type, the file shall be extracted as if it were a regular
 122 file. As this action occurs, the format-reading utility shall issue a warning to the standard
 123 error output.

124 ASCII digit '0' represents a regular file.

125 For backward compatibility, a *typeflag* value of binary zero (0) should be recognized as meaning a regular file when extracting
 126 files from the archive. Archives written with this version of the
 127 archive file format shall create regular files with a *typeflag* value
 128 of ASCII '0'.
 129

130 ASCII digit '1' represents a file linked to another file, of any type, previously
 131 archived. Such files are identified by each file having the same
 132 device and file serial number. The linked-to name is specified in
 133 the *linkname* field with a trailing null.

134 ASCII digit '2' is reserved.

135 ASCII digits '3' and '4' represent character special files and block special files
 136 respectively.

137 In this case the *devmajor* and *devminor* fields shall contain an
 138 encoding of the information found in the *st_rdev* field of the *stat*
 139 structure for the device file. Operating systems may map the
 140 device specifications to their own local specification, or may
 141 ignore the entry.

UNAPPROVED DRAFT. All Rights Reserved by IEEE.
 Do not specify or claim conformance to this document.

142 ASCII digit '5' specifies a directory or sub-directory. On systems where disk
143 allocation is performed on a directory basis the *size* field shall
144 contain the maximum number of bytes (which may be rounded to
145 the nearest disk block allocation unit) that the directory may hold.
146 A *size* field of zero indicates no such limiting. Systems that do
147 not support limiting in this manner should ignore the *size* field.

148 ASCII digit '6' specifies a FIFO special file. Note that the archiving of a FIFO file
149 archives the existence of this file and not its contents.

150 ASCII digit '7' is reserved.

151 ASCII letters 'A' through 'Z' are reserved for custom implementations. All other
152 values are reserved for specification in future revisions of the
153 standard.

154 The *magic* field is the specification that this archive was output in this archive format. If
155 this field contains TMAGIC, then the *uname* and *gname* fields shall contain the ASCII
156 representation of the owner and group of the file respectively. When the file is restored
157 by a privileged, protection-preserving version of the utility, the password and group files
158 shall be scanned for these names. If found, the user and group IDs contained within these
159 files shall be used rather than the values contained within the *uid* and *gid* fields. c

160 The encoding of the header is designed to be portable across machines.

161 D.1.1 References

162 <grp.h> §9.2.1, <pwd.h> §9.2.2, <sys/stat.h> §5.6.1, *stat()* §5.6.2, <unistd.h> §2.10. B

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

E. Alternative *wait()* Functions

1 It has been proposed that the following section replace Wait for Process Termination
2 §3.2.1. Consult the cover letter for the ballot associated with this draft for an explanation
3 of how to make your preferences known. Unless an explicit action is taken by the
4 Balloting Group, this section will not appear in the approved Full Use Standard.

5 E.1 Process Termination

6 ...

7 E.1.1 Wait for Process Termination

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

9 E.1.1.1 Synopsis

```
10         int wait(stat_loc)  
11         int *stat_loc;  
12  
13         int waitpid (stat_loc, pid, options)  
14         int *stat_loc;  
15         int pid;  
16         int options;
```

17 E.1.1.2 Description

18 The header `<sys/wait.h>` defines the following arguments for the *waitpid()* function:

<u>Constant</u>	<u>Description (<i>waitpid()</i> only)</u>
WNOHANG	return immediately if no children to wait for
WUNTRACED	also return status for stopped children if the implementation supports the Job Control Option

22

23 If *stat_loc* is not (int *) 0, information called *status* shall be stored in the location pointed
24 to by *stat_loc* as follows:

25 If the child process terminated due to an `_exit()` function, the low order 8 bits of
 26 *status* (corresponding to the octal value 0377) shall be zero, and the 8 bits
 27 corresponding to the octal value 0177400 shall contain the low order 8 bits of the
 28 argument that the child process passed to `_exit()` (see `_exit()` §3.2.2).

29 If the child process terminated due to a signal that was not caught, the low order 6
 30 bits of *status* (corresponding to the octal value 077) shall contain the number of
 31 the signal that caused the termination, and the 8 bits corresponding to the octal
 32 value 0177400 shall be zero. In addition, if the bit that would be masked by the
 33 octal value 0200 is set, an abnormal termination with actions occurred (see
 34 `sigaction()` §3.3.4).

35 If the `wait()` function returned due to an implementation defined condition, the bit
 36 of *status* corresponding to the octal value 0100 shall be set. The value of the
 37 other bits of *status* are implementation defined and the child may not have
 38 terminated. If the child has terminated, a subsequent `wait()` function shall return
 39 its *status*.

40 If a parent process terminates without waiting for its child processes to terminate, its
 41 children shall be assigned a new parent process ID corresponding to an implementation B
 42 defined system process. The `wait()` function shall only return successfully on the
 43 termination of a child process or due to an implementation defined change in status of a
 44 child process.

45 If the `waitpid()` variant is used, then the arguments *pid* and *options* are used to modify C
 46 the behavior of the function. C

47 If the *pid* argument specifies the child process for which *status* information is to be C
 48 obtained, the process determined by *pid* is determined as follows: C

49 > 0 The *pid* specifies the process ID of a child process. C

50 0 The *pid* specifies any single child process whose process group ID is equal C
 51 to that of the calling process. C

52 -1 The *pid* specifies any single child process. C

53 < -1 The *pid* specifies any single child process whose process group ID is equal C
 54 to the absolute value of *pid*. The absolute value of *pid* shall not exceed C
 55 {PID_MAX}. C

56 The *options* argument contains two options that may be combined by forming their C
 57 bitwise inclusive OR. C

58 If the *options* bit indicated by WNOHANG is set, then `waitpid()` will not suspend the C
 59 calling process if the process specified by *pid* has not terminated. If the implementation C
 60 supports the Job Control Option, then the calling process specified by *pid* has not been C
 61 stopped. In either case, a value of zero is returned by `waitpid()`. C

62 If the *options* bit indicated by WUNTRACED is set and if the implementation supports the c
63 Job Control Option, then *waitpid()* shall also return in *stat_loc* the wait status c
64 information when the process specified by *pid* is stopped due to a SIGTIN, SIGTOU, c
65 SIGTSTP, or SIGSTOP signal. In this case, the wait status information can also be c
66 interpreted in the following way: c

67 If the child process stopped, the 8 bits of *status* (corresponding to the octal value 8
68 0177400) shall contain the number of the signal that caused the process to stop 8
69 and the low order 8 bits corresponding to the octal value 0377 shall be set equal 8
70 to the octal value 0177. 8

71 If the implementation does not support the Job Control Option, then the WUNTRACED c
72 flag is ignored. c

73 E.1.1.3 Returns

74 If the *wait()* function returns due to the receipt of a signal by the calling process, a value
75 of -1 shall be returned to the calling process and *errno* shall be set to [EINTR]. If the
76 *wait()* function returns due to a terminated child process, the process ID of the child shall
77 be returned to the calling process. Otherwise, a value of -1 shall be returned, and *errno*
78 shall be set to indicate the error.

79 If the *waitpid()* function returns due to the termination of a process specified by *pid*, the c
80 process ID of the terminated child shall be returned to the calling process. c

81 If the implementation supports the Job Control Option and the *waitpid()* function is c
82 called with the WUNTRACED option, and the *waitpid()* function returns due to a process c
83 specified by *pid* having been stopped, the process ID of the stopped child shall be c
84 returned to the calling process. c

85 If *waitpid()* is called and the WNOHANG option is used, then a value of zero shall be c
86 returned for one of two reasons: c

87 1. The implementation supports the Job Control Option and the WUNTRACED option c
88 was used and the process specified by *pid* has not been stopped. c

89 2. The process specified by *pid* has not been terminated. c

90 Otherwise, the *waitpid()* function shall return a value of -1 and *errno* shall be set to c
91 indicate the error. c

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

92 E.1.1.4 Errors

93 If any of the following conditions occur, the *wait()* and *waitpid()* functions shall return c
 94 -1 and set *errno* to the corresponding value: c

95 [ECHILD] The calling process has no existing unwaited-for child processes.

96 [EINTR] The *wait()* function was terminated by a signal. The value pointed c
 97 to by *stat_loc* may be undefined. c

98 If any of the following conditions occur, the *waitpid()* function shall return -1 and set c
 99 *errno* to the corresponding value: c

100 [ECHILD] The process specified by *pid* is not a child process or does not c
 101 exist. c

102 [EINTR] The *waitpid()* function was terminated by a signal. The value c
 103 pointed to by the *stat_loc* may be undefined. c

104 [EINVAL] The *waitpid()* was called with an invalid *options* value. c

105 B

106 E.1.1.5 References

107 *exec* §3.1.2, *_exit()* §3.2.2, *fork()* §3.1.1, *pause()* §3.4.2, *times()* §4.5.2, *sigaction()* 8
 108 §3.3.4.

Identifier Index

<i>access()</i>	File Accessibility {5.6.3}	110
<i>alarm()</i>	Process Alarm Clock {3.4.1}	70
<i>asctime()</i>	Extensions to <i>asctime()</i> Function {8.1.1}	156
<i>chdir()</i>	Change Current Working Directory {5.2.1}	90
<i>chmod()</i>	Change File Modes {5.6.4}	111
<i>chown()</i>	Change Owner and Group of a File {5.6.5}	112
<i>close()</i>	Close a File {6.3.1}	121
<i>closedir()</i>	Directory Operations {5.1.2}	88
<i>cpio</i>	<i>cpio</i> Archive Format {10.1.1}	169
<i>creat()</i>	Create a New File or Rewrite an Existing One {5.3.2}	95
<i>ctermid()</i>	Generate Terminal Pathname {4.7.1}	84
<i>cuserid()</i>	Get User Name {4.2.4}	76
<i>directory</i>	Directory Operations {5.1.2}	88
< <i>dirent.h</i> >	Format of Directory Entries {5.1.1}	87
<i>dup()</i>	Duplicate an Open File Descriptor {6.2.1}	120
<i>dup2()</i>	Duplicate an Open File Descriptor {6.2.1}	120
<i>endgrent()</i>	Group Database Access {9.2.1}	166
<i>endpwent()</i>	User Database Access {9.2.2}	167
<i>environ</i>	Execute a File {3.1.2}	49
<i>errno</i>	Error Numbers {2.5}	32
< <i>errno.h</i> >	Error Numbers {2.5}	32
<i>exec</i>	Execute a File {3.1.2}	49
<i>execl()</i>	Execute a File {3.1.2}	49
<i>execle()</i>	Execute a File {3.1.2}	49
<i>execlp()</i>	Execute a File {3.1.2}	49
<i>execv()</i>	Execute a File {3.1.2}	49
<i>execve()</i>	Execute a File {3.1.2}	49
<i>execvp()</i>	Execute a File {3.1.2}	49
<i>_exit()</i>	Terminate a Process {3.2.2}	55
<i>fcntl()</i>	File Control {6.5.2}	128
< <i>fcntl.h</i> >	Data Definitions for File Control Operations {6.5.1}	127
<i>fdopen()</i>	Open a Stream on a File Descriptor {8.2.2}	161
<i>fileno()</i>	Map a Stream Pointer to a File Descriptor {8.2.1}	160
<i>fork()</i>	Process Creation {3.1.1}	47
<i>fpathconf()</i>	Get Configurable Pathname Variables {5.7.1}	116
<i>fstat()</i>	Get File Status {5.6.2}	108
<i>getcwd()</i>	Working Directory Pathname {5.2.2}	91
<i>getegid()</i>	Get Real User, Effective User, Real Group, and Effective Group IDs {4.2.1}	73
<i>getenv()</i>	Environment Access {4.6.1}	83
<i>geteuid()</i>	Get Real User, Effective User, Real Group, and Effective Group IDs {4.2.1}	73

<i>getgid()</i>	Get Real User, Effective User, Real Group, and Effective Group IDs {4.2.1}	73
<i>getgrent()</i>	Group Database Access {9.2.1}	166
<i>getgrgid()</i>	Group Database Access {9.2.1}	166
<i>getgrnam()</i>	Group Database Access {9.2.1}	166
<i>getgroups()</i>	Get Supplementary Group IDs {4.2.3}	75
<i>getlogin()</i>	Get User Name {4.2.4}	76
<i>getpgrp()</i>	Get Process Group ID {4.3.1}	78
<i>getpid()</i>	Get Process and Parent Process IDs {4.1.1}	73
<i>getppid()</i>	Get Process and Parent Process IDs {4.1.1}	73
<i>getpwent()</i>	User Database Access {9.2.2}	167
<i>getpwnam()</i>	User Database Access {9.2.2}	167
<i>getpwuid()</i>	User Database Access {9.2.2}	167
<i>getuid()</i>	Get Real User, Effective User, Real Group, and Effective Group IDs {4.2.1}	73
<grp.h>	Group Database Access {9.2.1}	166
<i>isatty()</i>	Determine Terminal Device Name {4.7.2}	85
<i>jcsetpgrp()</i>	Set Process Group ID for Job Control {4.3.3}	79
<i>kill()</i>	Send a Signal to a Process {3.3.2}	62
<limits.h>	Numerical Limits {2.9}	39
<i>link()</i>	Link to a File {5.3.4}	96
<i>longjmp()</i>	Non-Local Jumps {8.3.1}	162
<i>lseek()</i>	Reposition Read/Write File Offset {6.5.3}	133
<i>main()</i>	Execute a File {3.1.2}	49
<i>mkdir()</i>	Make a Directory {5.4.1}	97
<i>mkfifo()</i>	Make a FIFO Special File {5.4.2}	99
<i>open()</i>	Open a File {5.3.1}	92
<i>opendir()</i>	Directory Operations {5.1.2}	88
<i>pathconf()</i>	Get Configurable Pathname Variables {5.7.1}	116
<i>pause()</i>	Suspend Process Execution {3.4.2}	71
<i>pipe()</i>	Create an Inter-Process Channel {6.1.1}	119
<pwd.h>	User Database Access {9.2.2}	167
<i>read()</i>	Read from a File {6.4.1}	122
<i>readdir()</i>	Directory Operations {5.1.2}	88
<i>rename()</i>	Rename a File {5.5.3}	103
<i>rewinddir()</i>	Directory Operations {5.1.2}	88
<i>rmdir()</i>	Remove a Directory {5.5.2}	102
<i>setgid()</i>	Set User and Group IDs {4.2.2}	74
<i>setgrent()</i>	Group Database Access {9.2.1}	166
<i>setjmp()</i>	Non-Local Jumps {8.3.1}	162
<i>setlocale()</i>	Extensions to <i>setlocale()</i> Function {8.1.2}	158
<i>setpgrp()</i>	Set Process Group ID {4.3.2}	78
<i>setpwent()</i>	User Database Access {9.2.2}	167
<i>setuid()</i>	Set User and Group IDs {4.2.2}	74
<i>sigaction()</i>	Examine and Change Signal Action {3.3.4}	65

<i>sigaddset()</i>	Manipulate Signal Sets {3.3.3}	64
<i>sigdelset()</i>	Manipulate Signal Sets {3.3.3}	64
<i>sigfillset()</i>	Manipulate Signal Sets {3.3.3}	64
<i>siginitset()</i>	Manipulate Signal Sets {3.3.3}	64
<i>sigismember()</i>	Manipulate Signal Sets {3.3.3}	64
<i>siglongjmp()</i>	Non-Local Jumps {8.3.1}	162
<i>signal()</i>	Specify Signal Handling {8.3.2}	163
<signal.h>	Signal Names {3.3.1}	57
<i>sigpending()</i>	Examine Pending Signals {3.3.6}	68
<i>sigprocmask()</i>	Examine and Change Blocked Signals {3.3.5}	67
<i>sigsetjmp()</i>	Non-Local Jumps {8.3.1}	162
<i>sigsetops</i>	Manipulate Signal Sets {3.3.3}	64
<i>sigsuspend()</i>	Wait for a Signal {3.3.7}	69
<i>sleep()</i>	Delay Process Execution {3.4.3}	72
<i>stat()</i>	Get File Status {5.6.2}	108
<stat.h>	File Characteristics: Header File and Data Structure {5.6.1}	106
<i>sysconf()</i>	Get Configurable System Variables {4.8.1}	85
<sys/stat.h>	File Characteristics: Header File and Data Structure {5.6.1}	106
<sys/types.h>	Primitive System Data Types {2.6}	37
<sys/wait.h>	Wait for Process Termination {3.2.1}	53
<i>tar</i>	Extended <i>tar</i> Format {D.1}	285
<i>tcdrain()</i>	Line Control Functions {7.2.2}	151
<i>tcflow()</i>	Line Control Functions {7.2.2}	151
<i>tcflush()</i>	Line Control Functions {7.2.2}	151
<i>tcgetattr()</i>	Get and Set State {7.2.1}	149
<i>tcgetpgrp()</i>	Get Distinguished Process Group ID {7.2.3}	153
<i>tcsendbreak()</i>	Line Control Functions {7.2.2}	151
<i>tcsetattr()</i>	Get and Set State {7.2.1}	149
<i>tcsetpgrp()</i>	Set Distinguished Process Group ID {7.2.4}	154
<i>termios</i>	General Terminal Interface {7.1}	135
<termios.h>	Settable Parameters {7.1.2}	142
<i>time()</i>	Get System Time {4.5.1}	81
<i>times()</i>	Process Times {4.5.2}	82
<i>ttyname()</i>	Determine Terminal Device Name {4.7.2}	85
<types.h>	Primitive System Data Types {2.6}	37
<i>umask()</i>	Set File Creation Mask {5.3.3}	95
<i>uname()</i>	System Name {4.4.1}	80
<unistd.h>	Symbolic Constants {2.10}	43
<i>unlink()</i>	Remove Directory Entries {5.5.1}	100
<i>utime()</i>	Set File Access and Modification Times {5.6.6}	114
<utname.h>	System Name {4.4.1}	80
<i>wait</i>	Wait for Process Termination {3.2.1}	53
<i>wait2()</i>	Wait for Process Termination {3.2.1}	53
<wait.h>	Wait for Process Termination {3.2.1}	53
<i>write()</i>	Write to a File {6.4.2}	124

Topical Index

- /dev/console ... 275
- /dev/null ... 275
- /dev/tty ... 222, 231, 275
- /usr/group ... 4, 7-8, 178, 181-182, 186-190,
192-194, 220, 230, 237-238, 241-
242, 247-248, 250, 268, 273
- 1003 ... 181, 244
- 1003.1 ... 3-4, 7, 20-21, 27, 62, 155-156,
158, 172, 175-176, 178, 181-182,
196, 262, 273-284
- 1003.2 ... 176, 262
- 1003.4 ... 262
- 4.2 ... 4, 155, 189-190, 219-223, 225-226,
228-229, 233, 237, 244, 249-250,
255-257, 262
- 4.3 ... 4, 155, 188-190, 201, 203, 205, 209,
211, 218, 220-221, 223-225, 228-
230, 233, 237-238, 240, 242, 244,
247, 253, 255, 260, 262, 267, 270
- 8-bit characters ... 287
- ability ... 155, 212-213, 228, 258, 265
- abnormal termination ... 52-53, 58-59, 292
- abnormal termination with actions ... 52-53,
59, 292
- abort ... 52, 58, 155, 227
- absolute pathname ... 31-32, 91, 203, 205
- absolute value ... 63, 292
- abstract ... 200
- access ... 22-26, 29-31, 33, 37, 43, 52, 59,
61, 83-84, 87, 92, 94, 103, 106,
108-111, 114-115, 119, 128-131,
136, 165-167, 169-170, 176, 201-
205, 207, 215, 221-222, 224, 228,
231, 233, 236, 239-240, 247-249,
253, 262, 274, 281, 287
- access control ... 22, 25, 30, 59, 136, 205,
221-222, 253
- access mechanism ... 30-31, 169, 202, 204,
233, 248-249
- access modes ... 22, 26, 92, 111, 128-129,
131, 249
- access permissions ... 23-26, 29-31, 37, 52,
103, 110-111, 169-170, 202, 204,
239
- access time ... 31, 106, 114-115, 240, 249
- acknowledge ... 202, 210
- acos ... 155
- actime ... 114, 240
- action ... 29, 31, 44-45, 50, 52-53, 58-61,
65-67, 69, 71, 103, 136, 151-152,
163, 198, 200, 208, 218, 221-225,
238, 244, 247, 263-264, 267, 285,
288, 291-292
- activity ... 59, 72, 77, 119, 139, 244
- Ada ... 5, 178
- address ... 4-5, 21-22, 27-28, 34-35, 61, 77,
84, 176, 178-179, 188, 192, 200-
201, 203, 208, 211, 216, 218, 221-
222, 224, 235, 241, 255, 262, 276
- address space ... 21-22, 27-28, 235
- advantage ... 44, 232, 235, 240, 252
- advisory locks ... 248-249
- affected process ... 29, 50, 218, 222, 229,
248
- alarm ... 48, 51-52, 58, 61, 70-72, 226-227,
249
- alarm clock ... 51, 70-71, 226
- alarm requests ... 70-72
- allowed extensions ... 200, 202, 213, 242
- American National Standard ... 190
- amode ... 110-111, 281
- ANSI ... 21, 34, 36, 39-40, 52, 56, 58, 62,
108, 155-156, 158, 160-163, 175-
176, 178-179, 184, 186, 188, 190-
191, 196, 210, 239, 256-257, 259,
261, 266, 268, 274
- append ... 125, 128, 218, 260
- appendices ... 3, 175, 194-195
- appendix ... 3, 175, 181, 189, 194-196, 202,
215, 218, 233, 262, 267, 273-274
- application ... 3, 17-23, 25, 38, 41, 43-45,
56, 58, 60-61, 64, 86, 92, 101, 116-

- 117, 139, 143, 155, 165, 176-177, 179, 181-185, 192-193, 196-201, 204-209, 211-218, 220-226, 228, 230-237, 240-252, 258, 260-261, 267, 271, 279, 285
- Application Conformance ... 21, 198, 217
- application developer ... 17-18, 155, 185, 197, 205-206, 215, 226, 232, 252, 258
- application implementor ... 3, 226, 230, 252
- application portability ... 3, 17, 38, 44, 155, 182-183, 206, 209, 221-222
- application programs ... 18-19, 38, 56, 155, 176-177, 181, 183, 198-199, 206, 215, 233, 236, 252, 261, 271
- application requirements ... 19, 21, 197-198, 232
- application writer ... 155, 193, 211, 213, 226, 232, 236, 241, 243-247, 250-251, 285
- applications requiring ... 19-21, 176, 193, 197, 200, 206, 208-209, 211-213, 216-217, 224-225, 232, 237, 242, 244, 246, 249
- appropriate ... 5, 19, 22, 27, 30-31, 35-36, 45, 52, 59, 62, 74-75, 97, 101, 110-115, 117, 188, 193, 195, 200, 204, 209, 223-224, 226, 228, 235, 239-241, 247-248, 252-253, 263
- appropriate privileges ... 22, 27, 30, 36, 74-75, 97, 101, 110-114, 117, 200, 204
- appropriate signals ... 59, 223-224, 253
- appropriate value ... 117, 241
- arbitrary amount due ... 72
- arbitrary point ... 83
- archive ... 169-172, 190, 263-265, 267, 285, 287-289
- archive format ... 169, 172, 190, 263-264, 267, 285, 288
- archive tape ... 263, 285
- archive/interchange format ... 169, 262
- archiving files ... 169-172, 263-264, 267, 285, 288-289
- AREGTYPE ... 286
- arg list ... 33
- arg0 ... 49-50, 195, 217
- argc ... 49, 217, 278
- argn ... 49-50
- argument ... 21, 33-34, 49-51, 53-54, 61, 64-69, 75-76, 79-81, 86, 89-96, 98-104, 108-116, 121-122, 124, 126, 129-130, 132-134, 145, 147, 150, 152-154, 158, 161-162, 169, 177, 193-194, 217-219, 226, 235-236, 239, 243, 249-252, 258, 276, 278-279, 283, 292
- argument count ... 49
- argument list ... 33, 50-51, 86, 108, 116, 217, 226
- argv ... 49-50, 217, 278
- ARG_MAX ... 33, 39, 42, 50-51, 86, 217
- array ... 37, 39, 49-50, 76-77, 80, 84, 87, 91, 138, 142, 148, 198, 222, 229, 231, 234, 259, 287
- array size ... 87, 198, 234, 287
- ASCII ... 141, 156, 204, 261, 287-289
- asctime ... 38, 156, 256
- asin ... 155
- assert ... 155
- associated process group ... 23-26, 76, 135-136, 140-141, 153-154, 229, 255, 278, 283, 291
- associates ... 17, 22-26, 28, 30-32, 37, 50, 60, 66, 76-77, 85, 111, 116, 121-122, 124-125, 129, 131, 133-137, 140-141, 147, 149-150, 152-154, 160-161, 163, 187, 191, 198-200, 210-211, 217, 229, 231-232, 235, 248, 255, 260, 262, 278, 280, 282-283, 285, 291
- associating process group ... 23-26, 76, 135-136, 140-141, 153-154, 229, 255, 278, 283, 291
- assumptions ... 245
- asynchronous serial connection ... 146
- asynchronous terminal ... 146, 250
- AT&T ... 4, 182, 184, 188, 201, 203, 237, 241-242, 251, 270
- atan2 ... 155
- atexit ... 193, 219
- atof ... 155
- atoi ... 155, 199-200
- atol ... 155
- atomic ... 43, 216, 238, 244-246, 248
- atomicity ... 245
- attempt ... 4, 33-36, 59, 61, 67, 119, 122-126, 131, 136, 146, 149, 182-184, 195, 204, 211, 213-214, 222, 224, 238, 242, 244, 250, 252-253, 265, 287
- automatic ... 233
- availability ... 21
- available values ... 42, 50, 119, 122, 129, 232, 245
- avoid ... 143, 184, 200, 210, 223-224, 226-227, 231, 236, 242-243, 249, 252, 260-261, 263
- avoided conflict ... 223, 249, 260
- background ... 22, 59, 136, 147, 149, 153, 194, 219, 253

- background process ... 22, 136, 149, 153, 219, 253
- backward compatibility ... 242, 264, 267, 288
- Balloting Group ... 7-8, 186, 194, 262, 285, 291
- base document ... 4, 183, 188-189, 191-192, 202, 205
- Basic ... 178
- basic terminal input control ... 142
- baud ... 145-146
- beginners ... 271
- beginning ... 27-28, 30, 32, 70, 89-90, 92, 131, 133, 138, 140, 166-168, 195, 226, 254
- behavior ... 19-22, 54, 61-62, 91, 93, 137-138, 157, 159, 163, 191, 193, 207, 216-219, 223, 226-228, 237-238, 247, 257-258, 260, 263-264, 266, 273, 292
- believe ... 186
- Berkeley ... 188-189, 237, 240-241, 250, 270-271
- Bibliographic Notes ... 188, 200, 256, 268
- binary ... 18, 156, 184, 192, 214, 225, 232, 240, 260, 264, 285, 288
- binary compatibility ... 184, 225, 264, 288
- binary zero ... 285, 288
- binding ... 3, 17, 21, 155, 191, 199, 256
- bits ... 24-25, 30-31, 50, 53-55, 62, 66, 93, 95, 98-99, 107-108, 111-113, 129, 143-144, 146, 151, 209, 212, 218, 220, 230-231, 238, 240, 243, 249, 263, 278, 281, 284, 287-288, 292-293
- BLKTYPE ... 286
- block ... 22, 24, 60, 63-64, 66-69, 93, 106-107, 123, 125, 127, 130-131, 135-139, 148-149, 169, 171-172, 223-224, 226-227, 241-242, 245-246, 248, 253, 276, 280, 285-289
- block special ... 22, 24, 93, 106-107, 171-172, 280, 286, 288
- block special file ... 22, 24, 93, 107, 171, 280, 286, 288
- blocked ... 22, 24, 60, 63-64, 66-69, 93, 106-107, 123, 125, 127, 130-131, 135-139, 148-149, 169, 171-172, 223-224, 226-227, 241-242, 245-246, 248, 253, 276, 280, 285-289
- blocked signal ... 60, 63-64, 66-69, 136, 148-149, 223-224, 226-227, 253
- blocking ... 22, 24, 60, 63-64, 66-69, 93, 106-107, 123, 125, 127, 130-131, 135-139, 148-149, 169, 171-172, 223-224, 226-227, 241-242, 245-246, 248, 253, 276, 280, 285-289
- blocks ... 22, 24, 60, 63-64, 66-69, 93, 106-107, 123, 125, 127, 130-131, 135-139, 148-149, 169, 171-172, 223-224, 226-227, 241-242, 245-246, 248, 253, 276, 280, 285-289
- Bourne ... 210
- break ... 142-143, 151, 198, 212, 214, 257
- brk ... 193
- BRKINT ... 142
- broadly implementable ... 184, 191, 205
- BSD ... 4, 188-190, 201, 203, 205, 209, 211, 218-226, 228-230, 233, 237-238, 240-242, 244, 247, 249-250, 253, 255-257, 260, 262, 267, 270
- bsearch ... 155, 192
- buf ... 91, 108-109, 122-124, 235, 245
- buffer ... 82, 122-124, 140, 198, 219, 235, 240
- bug ... 233
- bus ... 178
- business ... 232
- byte ... 24, 27-28, 32-33, 39, 41-43, 50-51, 91, 122-126, 130-131, 133, 137-139, 141, 143, 156, 158, 170-171, 173, 206, 212, 229-231, 240-241, 243-244, 246, 248, 257, 264-265, 285-286, 288-289
- byte-oriented ... 169-170
- call ... 22, 25, 27-29, 32-34, 37, 44-45, 47-51, 53-56, 61-64, 66-74, 76-79, 82, 84, 86, 88-89, 95, 97, 101, 111-113, 115, 120, 129, 131, 136, 138, 140, 146, 149, 153-154, 162-163, 166, 168, 177, 183, 194, 199-200, 203-204, 207-208, 211, 216-220, 222-224, 226-229, 234, 238-239, 241-244, 247-248, 250, 252-253, 255, 258-260, 278-279, 287, 291-294
- callable function ... 62, 200
- caller ... 56, 72, 229, 255
- calling ... 22, 25, 27-29, 32-34, 37, 44-45, 47-51, 53-56, 61-64, 66-74, 76-79, 82, 84, 86, 88-89, 95, 97, 101, 111-113, 115, 120, 129, 131, 136, 138, 140, 146, 149, 153-154, 162-163, 166, 168, 177, 183, 194, 199-200, 203-204, 207-208, 211, 216-220, 222-224, 226-229, 234, 238-239, 241-244, 247-248, 250, 252-253, 255, 258-260, 278-279, 287, 291-294
- calling application ... 61, 64, 177, 216-217, 220, 223-224, 234, 241, 244, 250,

- 258
- calling process ... 22, 25, 27-29, 32, 34, 37, 44-45, 47-51, 53-56, 61, 63, 66-74, 76, 78-79, 82, 86, 97, 101, 111-113, 115, 131, 136, 149, 153-154, 166, 168, 216, 218-219, 222, 226-229, 238-239, 241, 244, 248, 253, 255, 278, 292-294
- calloc ... 155
- canonical mode ... 42, 137-138, 147-148, 253
- canonical mode input processing ... 42, 137-138, 147, 253
- capabilities ... 20, 38, 209, 228, 252
- capability ... 184, 189, 228, 235-236
- case ... 3, 17, 20, 32, 54, 60, 63, 72, 76, 103, 113, 138-139, 143, 158-159, 163, 179, 182, 184, 186, 188, 199-200, 203, 205-207, 222-223, 227-230, 233-235, 237, 239-240, 243-244, 246-249, 252-254, 264-265, 273, 276-277, 282, 288, 292-293
- CASE Services ... 179
- catch ... 61, 72, 208, 220-221, 223-224, 227, 261
- category ... 158-159, 193, 210, 258-259
- caught signal ... 34, 50, 53, 58-59, 66-67, 70-72, 121, 130, 141, 224, 253, 278, 292
- cause ... 23, 28, 34, 45, 47, 53-54, 59-60, 63, 67-70, 72, 89-90, 109, 111, 125, 131, 135-136, 141, 143, 147-149, 157, 187, 198, 205, 220, 223-224, 227, 241, 245, 248-250, 265-266, 292-293
- CBEMA ... 175, 178-179
- CCITT ... 178-179
- certain ... 23-25, 28, 31, 38-40, 136, 140, 142, 184, 192-193, 201, 208, 215, 219, 229, 257, 261
- certain calls ... 136
- certain characters ... 24, 31, 140
- cfgetispeed ... 146
- cfgetospeed ... 145
- cfsetispeed ... 146
- cfsetospeed ... 145
- cf_getispeed ... 284
- cf_getospeed ... 284
- cf_setispeed ... 284
- cf_setospeed ... 284
- change ... 5-6, 20, 23, 28, 30-31, 45, 54, 65, 67, 83, 86, 90, 103, 106, 108, 111-113, 136, 144, 146, 149, 153, 157-158, 183, 185, 187, 189, 192, 197, 202, 206-210, 218-221, 225-226, 233-236, 238-242, 247, 254, 256-257, 259-261, 265, 267, 276, 278, 282-283, 292
- Change Current Working Directory ... 90, 235
- Change File Modes ... 111, 239
- Change Owner and Group of File ... 112, 240
- char ... 49, 76, 83-85, 87-88, 90-92, 96-97, 99-100, 102-103, 108, 110-112, 114, 116, 122, 124, 142, 158, 161, 166-167, 199, 234, 244-245, 258-259, 278, 286
- character ... 22-24, 27, 29, 31, 37-39, 46, 49-50, 58-59, 77, 80, 84, 87, 91, 93, 106-107, 135-144, 146-148, 155-156, 158, 161, 171-172, 190, 193, 203-206, 211, 213, 225, 230, 233-234, 252-254, 256, 258-259, 267, 274, 280, 283, 287-288
- character array ... 39, 49-50, 77, 80, 84, 87, 91, 234, 259, 287
- character framing error ... 142-143
- character pointers ... 49-50, 77, 84, 91, 259
- character representation ... 77
- character size ... 87, 144, 146, 234, 287
- character special ... 22, 24, 29, 46, 58-59, 93, 106-107, 135, 138, 140-141, 147-148, 171-172, 225, 254, 280, 283, 288
- character special file ... 22, 24, 29, 93, 107, 141, 148, 280, 288
- character string ... 37, 50, 84, 161, 233, 258-259, 274, 287
- CHAR_BIT ... 40
- CHAR_MAX ... 40
- CHAR_MIN ... 40, 211
- chdir ... 61, 90-91, 239, 287
- check ... 62, 110, 142-143, 147, 158-159, 207, 222, 225, 227, 233-234, 239, 253, 255, 265, 279
- child ... 23, 27, 33, 47-48, 52-56, 59, 61, 82, 131, 136, 216, 218, 224, 231, 277-278, 292-294
- child process ... 23, 27, 33, 47-48, 52-56, 59, 61, 82, 131, 136, 216, 218, 224, 231, 277-278, 292-294
- child processes ... 23, 27, 33, 47-48, 52-56, 59, 61, 82, 131, 136, 216, 218, 224, 231, 277-278, 292-294
- child times ... 82, 231
- children ... 53-55, 66, 82, 215, 219-220, 222-225, 291-292
- children terminates ... 54-55, 82, 219-220, 223-224, 292

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- CHILD_MAX ... 42, 86, 215
 chksum ... 286, 288
 chmod ... 30-31, 50, 52, 61, 96, 99-100, 108, 111, 113, 173, 277
 chown ... 45, 61, 108, 112-113, 228, 277, 281
 chroot ... 204
 CHRTYPE ... 286
 circumstances ... 32, 45, 146, 223
 class-specific functions ... 190, 194, 250
 classical ... 265
 clear ... 30, 48, 56, 78, 93, 95, 107, 111, 113, 119, 123, 125, 129-130, 135, 137, 146, 183, 207-208, 211, 216, 220, 234, 238, 240, 245-246, 281
 clear errno ... 207-208, 234
 clearenv ... 231
 clearerr ... 155
 CLK_TCK ... 23, 37, 40, 82-83, 86, 226, 230-232, 277
 CLOCAL ... 144, 146
 clock tick ... 23
 clock_t ... 37, 82-83, 193, 209, 230-231, 277, 280
 close ... 56, 61, 89, 95, 102, 120-123, 127, 129, 131, 133, 141, 144, 146, 166, 168, 193, 198, 214, 221, 243, 254-255, 266, 281
 close file ... 121, 127, 129, 131, 141, 198, 214, 243, 254
 closedir ... 88-89, 190
 closing ... 56, 61, 89, 95, 102, 120-123, 127, 129, 131, 133, 141, 146, 166, 168, 193, 198, 214, 221, 243, 254-255, 266, 281
 closing terminal device file ... 141, 254
 cmask ... 95
 cmd ... 127-129, 131-132
 COBOL ... 266
 Cobol ... 178
 code ... 17-18, 22; 55, 98, 102, 104, 176, 184-185, 191, 200, 204-206, 213-214, 216, 220, 223, 225-226, 233, 242-243, 251, 259, 270, 287
 collation ... 37
 column ... 212
 combined argument ... 50, 292
 command ... 17, 25, 29, 38, 120, 129-130, 135, 149, 177, 195, 219-220, 222-223, 225, 247, 250-252, 260, 263, 276, 284
 command interpreter ... 25, 29, 38, 135, 223
 Command Name ... 195
 commercial applications ... 18
 committee ... 4, 7, 176, 178, 181-182, 250, 268
 common ... 155, 176-177, 179, 183, 185-186, 188, 204, 206, 209, 216, 227, 236, 244, 247, 256, 273
 common usage ... 155, 204, 256
 common uses ... 155, 209, 216, 227
 comparisons ... 139, 205, 209, 211, 243, 273, 276
 compatibility ... 4, 172, 184, 206, 209-210, 225, 237, 241-243, 264, 267, 288
 compile time ... 25, 44, 211, 215, 230, 232
 compiler ... 25, 44, 191, 200, 211, 213, 215, 229-231, 260
 complete ... 4, 33, 137, 140, 146, 166, 168, 194, 202, 210-211, 226, 239, 245, 257, 266
 complication ... 232
 computer architecture ... 232
 concepts ... 3, 30, 185, 200-204, 206-207, 214, 244, 246, 251, 263
 concern ... 6, 109, 181-182, 184, 191-193, 199-200, 210, 223, 240, 250, 252, 261-262, 264
 concurrent writes ... 247
 condition ... 20, 32-34, 36, 53-54, 60, 69, 78, 81, 83-85, 132, 142-143, 146, 198, 208, 215, 219, 227-228, 237, 241-242, 248, 265-266, 280-283, 292
 configurable pathname variables ... 116, 216, 232-233, 240-241
 configurable system variables ... 85, 184, 216, 232-233, 240, 276
 configuration ... 44, 177, 198, 233, 276
 conflict ... 33, 163, 185, 194-195, 218, 221, 223, 225, 242, 249, 260
 conform ... 4-5, 18-21, 25, 27, 39, 41, 43-44, 156, 169, 172, 176, 184-185, 188, 192-193, 197-199, 201, 210, 212-215, 217, 221-223, 226, 237, 239, 241-244, 259
 conformance ... 4-5, 18-21, 25, 27, 39, 41, 43-44, 156, 169, 172, 176, 184-185, 188, 192-193, 197-199, 201, 210, 212-215, 217, 221-223, 226, 237, 239, 241-244, 259
 conformance documentation ... 20-21, 197
 conforming application ... 18-21, 41, 43-44, 176, 185, 193, 197-199, 201, 212-215, 217, 222-223, 226, 241-244
 Conforming Application Using Extensions ... 21, 197, 199, 212-214, 242
 Conforming Implementation ... 217
 Conforming Languages ... 21
 conforming program ... 18-19, 21, 156, 176, 184, 198, 217, 239, 259

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- conforming system ... 4-5, 20, 156, 169, 172, 185, 192, 215, 221-222, 226, 239, 242-244
- connection ... 23, 92, 135, 146
- consensus ... 185-187, 207, 221, 242, 265
- consequences ... 3, 37, 52, 55-56, 156, 181, 198, 202, 216, 219, 239
- consider ... 3, 27-28, 32, 39-40, 44, 62, 141, 173, 188-189, 191-193, 195, 201, 205-207, 210-211, 216-218, 221, 226, 231, 233, 235, 241, 243-244, 247, 260, 262-263, 265
- constants ... 21, 25, 39, 42-45, 53, 57-59, 66, 77, 84, 86, 110, 116, 127-128, 133, 135, 183, 195, 211, 215-216, 230, 232, 241, 286, 291
- constraining links ... 238
- construct ... 19, 31, 49, 177, 207, 211, 225
- construct name ... 207
- contain ... 4, 6, 20, 23-25, 32, 37-38, 45, 49-50, 53-54, 77, 80, 82, 85, 88, 94, 96-104, 106, 112, 114, 126, 158, 165-167, 170-171, 181, 194, 212-214, 217, 222, 233, 237, 267, 278, 280, 285, 287-289, 292-293
- contents ... 20, 84, 91, 98, 123, 156, 169, 171, 181, 189, 198, 274, 285
- context ... 33-34, 108, 141, 188-189, 200, 206, 233, 235
- contiguous files ... 262, 264, 267
- continue signal ... 59, 222, 224-225, 253
- continues ... 6, 25, 48, 59-60, 143, 220, 222, 224-225, 250, 253
- continuous stream ... 151
- control ... 22-23, 25-30, 35, 44, 47, 54, 56, 58-60, 63, 66, 70-71, 77-79, 84, 127-129, 131, 135-136, 138, 140-144, 146-149, 151, 153-154, 163, 177-178, 189, 191, 200, 205, 208, 218-226, 229-232, 236, 247-248, 250-255, 258, 265, 275, 278-279, 283-284, 291-293
- control character ... 136, 143, 147-148, 225, 253-254, 283
- control chars ... 142
- control functions ... 25, 27-29, 35, 54, 60, 63, 70-71, 78-79, 84, 136, 141, 147-149, 151, 153-154, 163, 218, 222, 229, 253, 255, 275, 284, 293
- Control Modes ... 144, 254
- Control Operations on Files ... 127, 247
- controlling process ... 22-23, 25-30, 47, 56, 60, 63, 70-71, 78-79, 84, 131, 135-136, 140-141, 146, 148, 153-154, 163, 219-220, 222, 224-225, 229-231, 248, 253, 278, 283, 292-293
- controlling terminal ... 22-23, 25-26, 30, 56, 58-59, 63, 77-79, 84, 136, 140-144, 146, 148-149, 153-154, 177, 200, 222, 225, 229, 231, 253, 255, 265, 275, 278, 283
- controversy ... 243
- CONTTYPE ... 286
- cooked mode ... 251
- Coordinated Universal Time ... 24, 202
- copies ... 91, 185, 259
- copyright ... 181
- core ... 217-218
- core file ... 217-218
- correspond ... 3, 25, 30, 36, 38, 42, 47, 49, 53-55, 64, 86, 88-89, 108, 111, 116-117, 129, 158-160, 189-190, 192, 195, 198, 201-202, 208, 212, 214-218, 228, 243-244, 258-259, 267, 292-293
- corresponding permissions ... 25, 30, 108, 111, 202
- corrupt ... 236
- cover ... 8, 200, 225, 228, 243, 274, 285, 291
- covert channel ... 225
- cpio ... 169, 172, 191, 262-265, 285
- CRDLY ... 284
- CREAD ... 144, 146
- creat ... 26-30, 39, 46, 48, 61, 92, 94-96, 98-99, 108-109, 119, 121-122, 124, 126, 128, 130-131, 134, 169, 184, 198, 212-213, 218-219, 228-229, 236-237, 239, 242, 247, 260, 262-266, 277, 280, 285, 287-288
- create ... 26-30, 39, 46, 48, 61, 92, 94-96, 98-99, 108-109, 119, 121-122, 124, 126, 128, 130-131, 134, 169, 184, 198, 212-213, 218-219, 228-229, 236-237, 239, 242, 247, 260, 262-266, 277, 280, 285, 287-288
- create inter-process channel ... 119, 242
- create new file ... 95-96, 99, 236
- created directory ... 46, 92, 96, 98-99, 218, 228, 280
- creating ... 26-30, 39, 46, 48, 61, 92, 94-96, 98-99, 108-109, 119, 121-122, 124, 126, 128, 130-131, 134, 169, 184, 198, 212-213, 218-219, 228-229, 236-237, 239, 242, 247, 260, 262-266, 277, 280, 285, 287-288
- creation ... 26, 28, 30, 47, 51, 87, 92-93, 95-99, 216-217, 236-237, 239, 244
- criterion ... 212
- CSIZE ... 144, 146

- CSMA/CD ... 178
- CSTOPB ... 144, 146
- ctermid ... 84, 192, 231
- ctime ... 156
- current directory ... 23, 30, 32, 38, 44, 51, 88-91, 102, 117, 210, 217-218, 235
- current file ... 31, 44, 49, 114, 117, 122, 124, 131, 217-218
- current operating system ... 80, 191
- current position ... 88-89, 122, 124, 131, 265
- current process ... 31-32, 49, 72, 77, 84, 136, 148, 162, 215, 217-218
- current process image ... 49, 217
- current state ... 89, 217
- current time ... 31, 44, 72, 114, 157, 257
- current user ... 66, 77, 215, 218
- current value ... 44, 86, 89, 116-117, 131, 133, 162, 259, 264
- current working directory ... 23, 30, 32, 38, 44, 51, 90-91, 102, 210, 217, 235
- cuserid ... 76-78, 168, 192, 274
- c_cc ... 138, 142, 148
- c_cflag ... 142, 144-146
- c_dev ... 170
- c_filedata ... 170-171
- c_filesize ... 170-171
- c_gid ... 170
- c_iflag ... 137, 142
- c_ino ... 170, 264
- C_IRGRP ... 172
- C_IROTH ... 172
- C_IRUSR ... 172
- C_ISBLK ... 172
- C_ISCHR ... 172
- C_ISDIR ... 172
- C_ISFIFO ... 172
- C_ISGID ... 172
- C_ISREG ... 172
- C_ISUID ... 172
- C_ISVTX ... 172
- C_IWGRP ... 172
- C_IWOTH ... 172
- C_IWUSR ... 172
- C_IXGRP ... 172
- C_IXOTH ... 172
- C_IXUSR ... 172
- c_lflag ... 137, 142, 147
- c_magic ... 170
- c_min ... 254
- c_mode ... 170, 172, 264
- c_mtime ... 170-171
- c_name ... 170-171
- c_namesize ... 170-171, 264
- c_nlink ... 170
- c_offlag ... 140, 142, 144
- c_rdev ... 170-171
- c_time ... 254
- c_uid ... 170
- Data Base Standards ... 179
- Data Definitions for File Control Operations ... 127, 247
- Data Interchange Format ... 262
- data items ... 80
- data losses ... 198, 247-248, 265
- data structure ... 31, 80, 82, 106, 171, 198, 204, 216, 222, 238
- Data Types ... 37, 195, 209, 277
- database ... 37-38, 77, 165-168, 177, 179, 248-249, 257, 261-262, 274
- database access ... 165-167, 262, 274
- Date and Time ... 156
- date/time ... 38, 258
- daylight ... 158, 256
- de facto standard ... 265
- deadlock ... 33, 131-132, 249
- decimal ... 157, 201, 286
- decision ... 25, 186, 194, 202, 219, 240
- declared type ... 22, 57, 225
- decrement ... 100
- default ... 50, 57-60, 136, 148, 152, 156, 158-159, 165, 167, 189, 205, 222, 224, 258-259
- default action ... 50, 59-60, 136, 224
- default shell ... 167
- default value ... 156, 158-159, 259
- define tversion ... 286
- Defined Terms ... 195
- defined type ... 22, 24, 37, 57, 64, 87-88, 106, 130, 161, 166, 170, 172, 202, 209, 220, 231, 277
- defines ... 5, 17-24, 26, 28-30, 32-34, 36-40, 43-46, 48, 50, 53-69, 74-78, 80-88, 106, 108-109, 111, 113-114, 116-117, 126-127, 129-131, 133, 135, 138, 142, 144, 148, 151, 155-156, 158-159, 161, 163, 166-167, 169-173, 176, 183-184, 191-197, 200-203, 205, 207-209, 211, 213-214, 217-220, 222, 224-225, 229-231, 233, 235, 237-239, 250-251, 254, 256-257, 261-263, 265, 268, 273-275, 277, 279, 281, 286, 288, 291-292
- definition ... 3, 17, 21-22, 25, 37, 39, 42, 92, 96, 100, 127, 135, 162, 183-185, 188, 191-198, 200, 204, 206, 210-211, 226, 233, 236, 238, 243, 247, 249, 251, 263, 273-275
- dclay ... 70, 72, 93, 124, 126, 128, 227, 282, 284

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- delay process execution ... 72, 227
- deliberations ... 181-182, 189
- deliver ... 48, 54, 59-61, 63-64, 66, 68-69, 71, 163, 216, 223-224
- delivery ... 48, 54, 59-61, 63-64, 66, 68-69, 71, 163, 216, 223-224
- Department of Defense ... 180
- depend ... 42, 45, 131, 137, 146, 151, 193, 197, 201, 206-208, 212-213, 216, 224, 230, 235-236, 240, 245, 256
- dependent ... 32, 96, 100, 137, 171, 223, 250
- descendant ... 29, 82, 103-104, 216, 218, 222, 225, 255
- describe ... 3-4, 17-18, 20-21, 23-25, 27-29, 31-33, 45, 47, 52, 66, 86, 88, 106, 108-111, 117, 129-130, 135-138, 142, 144, 147, 149, 156-157, 160, 165, 169, 172, 175, 180, 182-183, 189, 191, 201-205, 214, 219, 222-223, 227, 233-234, 238-240, 244, 247, 257-258, 263-265, 270, 273-274, 276, 285, 287
- described members ... 66, 106
- description ... 3, 5, 21, 24, 26, 33-34, 37-38, 41-44, 47, 49-50, 52-53, 55, 57-59, 62, 64, 66-67, 69-74, 76-88, 90-92, 95-96, 98-100, 102-103, 106, 108, 110-112, 114, 116, 119-122, 124, 127-130, 133, 135, 142, 144-145, 147-149, 151, 153-154, 156, 158, 160-163, 166-167, 193, 199-200, 203, 208, 210, 222, 234, 243, 247-248, 253, 263-264, 267, 291
- design ... 176, 183-184, 193, 240, 248, 267, 289
- designated structure ... 114
- desired time ... 70, 257
- desired way ... 224, 239
- detection ... 34, 58-59, 87, 143, 146, 249
- Determine Terminal Device Name ... 85, 231
- determining ... 4, 23, 29, 59, 137, 170-171, 197, 211, 232, 244, 253, 266, 292
- deterministic ... 265
- development ... 176-178, 181-183, 185, 197-198, 215, 274
- device ... 22-23, 29, 35, 37, 85, 93, 106, 126, 133, 135, 137, 140-141, 143-144, 150, 152-154, 179, 190, 194, 201, 203, 208, 213, 231, 241, 248, 250-251, 253-254, 275-276, 280, 282, 288
- device dependent ... 137, 250
- device number ... 37, 106, 143, 201, 213, 288
- Device- and Class-Specific Functions ... 190, 194, 250
- devmajor ... 286, 288
- devminor ... 286, 288
- dev_t ... 37, 106, 209, 238
- Diagnostics ... 155
- differ ... 21, 191, 201, 204, 225, 241, 243, 260, 273
- difference ... 24, 189, 192, 198, 220, 229, 231, 233-234, 251, 255, 273-274, 276-277
- differing implementations ... 201, 204, 225
- difficulties ... 194, 207, 209, 211, 250, 252, 254, 258
- digits ... 156-157, 209, 256, 287-289
- direct ... 3, 31, 48, 185, 188, 205, 216, 233, 258
- directories ... 31, 42, 49, 87, 97, 101, 104, 108, 171, 185, 198, 201, 233, 238
- directory ... 23-24, 26, 28, 30-32, 34-38, 44-46, 51-52, 87-94, 96-104, 106-107, 109-110, 112-113, 115-117, 165, 167, 172, 177, 184, 190, 193, 198, 201-205, 207, 210, 217-218, 228, 233-238, 240-241, 267, 275-277, 280, 286, 289
- directory access ... 24, 30, 103, 201, 233
- directory entry ... 23, 26, 88, 96, 98-99, 101, 103, 106, 201, 234, 236
- directory level ... 233, 240
- directory operations ... 88, 104, 184, 190, 201, 234-235
- directory routines ... 201
- directory search permission ... 51, 96, 98, 101-102, 107, 236
- directory stream ... 88-90, 235
- dirent ... 87, 89, 233-234
- dirent.h ... 87-88, 90, 233
- dirname ... 88-89
- DIRTYTYPE ... 286
- DIR_LEVEL_MAX ... 214
- disable ... 31, 46, 141, 143, 147
- disconnect ... 58, 141, 146, 254
- discussions ... 186, 189, 195, 220-221, 232, 239, 254
- distinction ... 156, 183, 192, 194, 204-205, 219, 247, 260
- distinguished process group ... 22, 25, 29, 135-136, 140-141, 148, 153-154, 221, 255, 283
- document ... 4, 6, 19-21, 48, 60, 68-69, 175-178, 180-181, 183, 185-192, 201-202, 205, 222-223, 239, 261-262, 264, 273
- documentation ... 3-4, 20-21, 48, 50, 60, 80,

- 183, 187, 189, 197
- domain ... 21, 33-34, 202, 219
- dominate ... 239
- donating ... 8
- dot ... 23-24, 26-27, 32, 35, 45, 88, 98, 102-104, 201, 210, 238
- dot-dot ... 23-24, 26, 32, 35, 45, 88, 98, 102-104, 201, 207, 238, 277
- double initial slash ... 207
- draft ... 6, 45, 155, 177-178, 181, 185, 187-190, 194, 209-210, 212, 214, 247, 257, 259, 285, 291
- drops ... 212, 223
- dup ... 27, 61, 95, 109, 120, 122, 124, 126, 134, 247, 260
- dup2 ... 61, 120-121, 243
- Duplicate an Open File Descriptor ... 120, 243
- Duplicate file descriptor ... 127
- duplicate open file descriptor ... 120, 243
- duration ... 66, 151
- d_name ... 87, 234
- e.g. ... 37, 108, 139, 141, 146, 158, 177, 185, 191, 195, 204, 208, 216, 221, 223, 236, 246, 251, 263-264
- E2BIG ... 33, 51, 217
- EACCES ... 33, 51, 89-91, 94, 96, 98, 100-102, 104, 109-110, 112-113, 115, 132, 236
- eaccess ... 239
- EAGAIN ... 33, 48, 123-126, 137, 241, 245-246, 282
- east ... 5, 156-157, 206, 256
- EBADF ... 33, 89-90, 109, 121-122, 124, 126, 132, 134, 150, 152-154, 242-243
- EBUSY ... 33, 101, 103, 105
- ECHILD ... 33, 55, 294
- ECHO ... 147
- ECHOE ... 147
- ECHOK ... 147
- ECHONL ... 147
- EDEADLK ... 33, 131-132
- editor ... 6-8, 45, 194, 238, 249
- EDOM ... 33
- EEXIST ... 34, 94, 96, 98, 100, 102, 104, 237, 280
- EFAULT ... 34, 208, 217
- EFBIG ... 34, 126
- effect ... 17, 26, 32, 35, 51, 60, 88-90, 92-94, 96-104, 109-115, 123, 136, 139, 141, 148, 158, 163, 166, 168, 216, 222, 224, 230, 239, 255-256, 282
- effective group ... 23-25, 28-29, 45-46, 50, 73-75, 92, 98-99, 107, 110-112, 227-228, 240, 278, 280
- effective user ... 23, 25, 29-30, 44-45, 50, 62, 73-74, 77, 92, 98-99, 107, 110-115, 225, 227-228, 239, 249, 278-279
- effective user ID ... 23, 25, 29-30, 44-45, 50, 62, 73-74, 77, 92, 98-99, 107, 110-115, 225, 227-228, 239, 278-279
- effort ... 4, 8, 17-18, 175-176, 178, 182, 244, 248
- EINTR ... 34, 54-55, 70-71, 94, 121-126, 130, 132, 152, 195, 208, 217, 220, 244, 281, 283, 293-294
- EINVAL ... 34, 55, 64-65, 67-68, 75-76, 79, 91, 104, 111, 113, 132, 134, 150, 152-154, 250, 281, 283, 294
- EIO ... 34, 136, 282
- EISDIR ... 34, 94, 104
- either routine ... 77, 168
- either type ... 22, 130, 234, 267, 277
- either value ... 30, 63, 110, 232, 237, 292
- elapsed real time ... 83
- element ... 82, 84, 87, 106, 148, 234, 249, 283
- ellipsis ... 225, 247
- EMFILE ... 34, 89, 94, 120-121, 132
- EMLINK ... 34, 96, 98
- empty directory entries ... 35, 45, 88, 98, 103
- empty pipe ... 123
- empty string ... 35, 39, 84, 90, 94, 97-98, 100-102, 104, 109-110, 112-113, 115, 159
- ENAMETOOLONG ... 34, 51, 90, 94, 97-98, 100-102, 104, 109-110, 112-113, 115, 208, 237-238, 276
- encode ... 204, 206, 261, 274, 288-289
- encoded ... 204, 206, 261, 274, 288-289
- encoded password ... 261
- encoded string ... 274
- end-of-archive ... 265, 285
- end-of-file ... 123, 138, 140-141, 146, 156, 173, 265-267
- end-of-medium ... 265-267
- endgrent ... 166, 274
- endpwent ... 167-168, 274
- ENFILE ... 35, 94, 120
- Enhanced Signals ... 275
- enhancements ... 267
- ENODEV ... 35, 248
- ENOENT ... 35, 52, 90, 94, 97-98, 100-102, 104, 109-110, 112-113, 115, 237, 276
- ENOEXEC ... 35, 52, 217

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- ENOLCK ... 35, 132, 248
- ENOMEM ... 35, 48, 52, 208
- ENOSPC ... 35, 94, 97-98, 100, 104, 126
- ENOTBLK ... 276
- ENOTDIR ... 35, 52, 89-90, 94, 97, 99-102, 104, 109-110, 112-113, 115
- ENOTEMPTY ... 35, 102, 104, 237-238, 276, 280
- ENOTTY ... 35, 79, 150, 152-154, 208
- entire database ... 166, 168, 257
- entries ... 23, 26, 31, 35, 45, 76, 87-88, 98, 100, 102-104, 108, 169, 171, 201, 233, 238, 264
- entry ... 23, 26, 60, 77, 88, 96, 98-99, 101, 103, 106, 166-171, 201, 234, 236, 239, 253, 280, 288
- environ ... 37, 39, 42, 49-50, 83, 217
- environment ... 3, 17, 20, 33, 37-39, 49-52, 83, 156, 158-159, 162, 177, 181, 183, 191-193, 195, 197, 201, 210-211, 217, 227, 231-232, 250, 254, 256-259, 274, 277
- environment access ... 83, 231
- Environment Description ... 37, 50, 52, 83, 193, 210
- environment list ... 33, 50-51, 83, 217
- environment strings ... 37, 50, 83, 158-159, 258-259
- environment variable names ... 38-39, 158, 210, 257
- environment variables ... 37-39, 49-50, 83, 156, 158-159, 177, 195, 210-211, 231, 256-258, 277
- envp ... 39, 49-50, 217, 278
- ENXIO ... 35, 94, 280, 282
- EOF ... 44, 130, 138, 140-141, 147-148, 254
- EOL ... 138, 140-141, 147-148, 254
- EPERM ... 36, 64, 75, 79, 97, 101, 112-113, 115, 154, 225
- EPIPE ... 36, 126
- Epoch ... 24, 81, 108, 114, 201-202, 256
- equal ... 27, 42, 44-45, 56, 63, 68, 74-76, 79, 91, 112, 121, 125, 129, 132, 154, 171, 213, 227-228, 243, 255, 278, 292-293
- equivalent ... 39, 95, 120, 146, 161, 193, 207, 219, 221, 233, 238
- ERANGE ... 36, 91
- ERASE ... 138, 140-141, 147-148, 254
- ERASE character ... 147-148, 254
- EROFS ... 36, 94, 97, 99-101, 103-104, 110, 112-113, 115
- errno ... 32, 48, 51, 54, 63, 65, 67-71, 75-76, 79, 81, 83, 89-91, 94, 96, 98-99, 101-102, 104, 109-111, 113, 115, 120-123, 125-126, 130-131, 133, 137, 150, 152-154, 193, 195, 204, 207-208, 220, 234, 237, 245-246, 248, 282, 293
- errno.h ... 32-33, 237
- error ... 3, 6, 17, 32-36, 46, 48, 51, 54-55, 62-63, 65, 67-71, 75-76, 78-79, 81, 83-85, 89-91, 93-94, 96, 98-102, 104, 109-113, 115, 120-124, 126, 131-136, 142-143, 150, 152-154, 160, 167-168, 183, 193, 195, 207-208, 216-217, 220, 223-224, 232, 234-237, 239, 241-248, 256, 265-266, 276, 279-283, 287-288, 293-294
- error number ... 32-33, 76, 183, 193, 195, 207-208, 244-245, 276
- Error Numbers ... 32, 193, 195, 207-208, 276
- Errors sections ... 6, 33
- ESPIPE ... 36, 134
- ESRCH ... 36, 64, 225
- establishes ... 23, 29, 61, 92, 130, 146, 163, 176, 224, 228
- ETXTBSY ... 217, 276, 280-281
- event ... 29, 54, 59, 77, 137-138, 146-147, 163, 184, 186, 191, 194, 203-206, 211-213, 216-217, 221, 224-225, 232-233, 239-240, 242-243, 245, 247, 257
- event notification ... 221
- ever-changing values ... 233
- evolution ... 269
- exactly ... 26, 92, 185, 192, 251, 265
- examine ... 65-68, 204, 220-221, 225-226, 257
- Examine and Change Blocked Signals ... 67, 226
- Examine and Change Signal Action ... 65, 221, 225
- Examine Pending Signals ... 68, 220, 226
- example ... 3, 29, 31, 34-35, 42, 59, 83, 122, 125, 139, 146, 184-185, 194-195, 198-202, 204-205, 213-215, 218, 222-224, 232, 240-241, 247-250, 252, 257-259, 263, 266
- exceed ... 34, 51, 63, 72, 75, 79, 87, 90, 94, 96-98, 100-102, 104, 109-110, 112-113, 115, 120-121, 126, 132, 137-138, 143, 154, 234, 244, 248, 276, 292
- exception ... 26, 29, 60, 125, 184, 186-187, 191, 194, 198, 208, 220-221, 233
- exceptional behavior ... 223
- excess ... 198

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- excluding ... 24, 63, 184, 186, 191, 195, 200, 204, 234, 237, 252, 263
- EXDEV ... 36, 97, 105
- exec ... 23, 28-29, 35, 39, 42, 45, 47-51; 55, 60-61, 66, 71, 73, 75, 78, 83, 95, 100, 107, 121-122, 127, 129, 133, 189, 193, 217-218, 226, 228, 239, 248, 260, 278, 294
- exec family ... 49, 129
- exec functions ... 47, 49, 51, 66, 127, 129, 218, 226, 239
- execl ... 23, 28-29, 35, 39, 42, 45, 47-51, 55, 60-61, 66, 71, 73, 75, 78, 83, 95, 100, 107, 121-122, 127, 129, 133, 189, 193, 217-218, 226, 228, 239, 248, 260, 278, 294
- execle ... 49, 226
- execlp ... 49-50, 226
- executable binary form ... 240
- executable form ... 232, 240
- execute access bits ... 30
- execute by group ... 172
- execute by others ... 172
- execute by owner ... 172
- execute file ... 30, 35, 49, 107-108, 110, 217, 239, 249, 280
- execute mode ... 249
- execute permission ... 30, 35, 43, 107-108, 110, 239
- executing instructions ... 82
- execution ... 25, 34, 44-45, 48, 51-53, 60-61, 66, 70-72, 107, 111, 127, 129, 177, 184, 192, 197, 211, 214, 216, 227, 232, 240, 287
- execution environment ... 192, 197, 211
- execution time ... 44-45, 72, 216, 232
- execv ... 23, 28-29, 35, 39, 42, 45, 47-51, 55, 60-61, 66, 71, 73, 75, 78, 83, 95, 100, 107, 121-122, 127, 129, 133, 189, 193, 217-218, 228, 239, 248, 260, 278, 294
- execve ... 49-50
- execvp ... 49-50
- existence ... 25, 28, 33-35, 43, 52, 55, 69, 90, 92-98, 100-101, 103-104, 109-110, 112-113, 115, 117, 128, 133, 139, 172, 183-185, 189-190, 193, 197, 200, 203, 206-208, 216-218, 220, 223-225, 229-230, 236-238, 241-243, 246, 248, 251, 256-257, 259, 261-264, 277, 280, 289, 294
- existing applications ... 139, 183-185, 206, 208, 216, 220, 224-225, 236-237, 241-243, 246, 251, 261
- existing data ... 133, 216, 262-264
- existing file ... 34, 43, 52, 92-97, 101, 103, 109-110, 112-113, 115, 117, 128, 133, 139, 172, 189, 217, 236, 238, 242-243, 246, 262-264, 280, 289
- existing implementations ... 25, 28, 117, 185, 197, 200, 206, 208, 216, 218, 220, 223-225, 230, 237, 241-242, 246, 248, 251, 257, 262
- existing mechanism ... 200, 220, 224-225, 248
- existing programs ... 139, 183, 206, 217-218, 223, 259, 261-262
- exists ... 25, 28, 33-35, 43, 52, 55, 69, 90, 92-98, 100-101, 103-104, 109-110, 112-113, 115, 117, 128, 133, 139, 172, 183-185, 189-190, 193, 197, 200, 203, 206-208, 216-218, 220, 223-225, 229-230, 236-238, 241-243, 246, 248, 251, 256-257, 259, 261-264, 277, 280, 289, 294
- exit ... 52, 55-56, 155, 193, 216, 219-220
- exit status code ... 55
- extend ... 4, 20-21, 35, 98, 100, 131, 133, 185, 190, 193, 210, 239, 250, 256-257, 262, 267, 275, 285, 287
- extended function ... 35, 133, 185, 239, 250, 275
- extended tar ... 190, 262, 285
- extension ... 4-5, 20-21, 32, 156, 158, 178, 197, 199-202, 212-214, 221, 226, 230, 239, 242, 256-257, 262-264, 267-268, 274
- external characteristics ... 18, 273
- external variable ... 32, 37, 50, 158, 191
- fabs ... 155
- facilities ... 3-5, 18, 20-21, 25, 44-45, 177, 184, 189, 191, 196-198, 200-201, 226, 232, 244
- facility ... 25, 176, 189, 207, 232-233, 240, 242, 244, 248-249, 252, 257, 267
- family ... 49, 129, 182, 214-215, 218
- fast bursts ... 254
- fast File System ... 240
- favor ... 195, 200, 207, 223
- FCHR_MAX ... 116, 212-213
- fclose ... 155, 193
- fcntl ... 27, 48, 50, 52, 61, 95, 109, 119-122, 124, 126-131, 134, 137, 243, 246-250, 253, 260, 283, 287
- fcntl.h ... 50, 92, 95, 127-130, 133, 283
- fdopen ... 161, 192, 260
- FD_CLOEXEC ... 50, 127, 129
- feature ... 19-20, 139, 176, 181, 183-184, 188, 198, 204, 216, 220-221, 232, 240-241, 247-248, 250, 252, 254,

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 263, 275
- Federal Information Processing Standards ... 180
- federal law ... 158, 256
- feof ... 155
- ferror ... 155
- few ... 184-185, 189, 191, 196, 223, 235, 250, 257
- fflush ... 155
- fgetc ... 155
- fgets ... 155
- field ... 31, 37, 51, 66-67, 89, 93-94, 96, 98-99, 101-102, 104, 106, 108-109, 111, 113-114, 119, 123, 126, 131, 142, 144-145, 147, 157, 163, 165, 169-170, 172, 186, 225, 233, 239, 261, 264, 267, 286-289
- FIFO ... 24, 27, 36, 46, 93-94, 99, 107, 121-123, 125-126, 134, 172, 189, 202-203, 228, 237, 244-246, 274, 286, 289
- FIFO special file ... 24, 27, 99, 107, 121-122, 189, 202, 286
- FIFOTYPE ... 286
- filides ... 85, 108-109, 116, 119-126, 128-129, 132-134, 149-154, 161, 242-243, 245
- filides2 ... 120-121, 243
- file ... 22-31, 33-38, 42-52, 55, 84-85, 87-89, 92-117, 119-137, 139-141, 143, 146, 148-150, 152-154, 160-161, 169-173, 177, 179, 189, 191-192, 195, 198, 200-207, 209, 211-219, 222, 228, 231, 233-234, 236-244, 246-250, 253-254, 256-257, 260-267, 274-277, 280-283, 285-289
- file access permissions ... 23-26, 29-31, 37, 52, 110-111, 169-170, 204, 239
- file accessibility ... 110, 239
- File Characteristics ... 106, 238
- file control ... 30, 35, 127-128, 136, 141, 148, 153-154, 191, 222, 231, 236, 247
- file control operations ... 35, 127, 247
- file creation mask ... 51, 93, 95-96, 98-99, 236
- file descriptor ... 24, 26, 33-34, 47, 50, 55, 85, 89, 92, 94, 103-109, 111, 116, 119-122, 124-127, 129-135, 149-150, 152-154, 160-161, 203, 222, 231, 239-241, 243, 247-248, 260, 282
- file descriptor deassignment ... 121, 243
- File Descriptor Manipulation ... 120, 243
- file hierarchy ... 31, 202, 204-205, 236, 263
- file mode ... 22, 24-26, 50-51, 92-93, 95-96, 98-99, 106-107, 111-112, 128-129, 131, 137, 139, 141, 146, 161, 169, 192, 239, 249, 260, 263-264, 287-288
- file mode creation mask ... 51, 93, 95-96, 98-99
- file name ... 23-25, 34, 42, 102-104, 110, 116-117, 169-171, 205-207, 218, 234, 236, 238, 241, 264-265, 267, 287-288
- file name length ... 42, 171, 234
- file named core ... 217-218
- file names ... 23-25, 34, 42, 102-104, 110, 116-117, 169-171, 205-207, 218, 234, 236, 238, 241, 264-265, 267, 287-288
- file offset ... 24, 26, 44, 92, 122, 124-125, 131, 133-134, 244, 247, 249-250
- file open ... 24, 26, 31, 33-35, 42, 47, 50, 55, 89, 92-95, 100, 108, 111, 116, 119-120, 122, 124, 126, 128-133, 135-136, 146, 149, 152, 161, 198, 203, 212, 214-217, 219, 222, 231, 236, 239-240, 243, 247-249, 253, 260
- file owners ... 24-25, 31, 36, 46, 50, 106-107, 111-115, 218, 240, 263, 287-289
- file permission ... 23-26, 29-31, 35, 37, 51-52, 92-95, 98-99, 107-108, 110-111, 114, 169-170, 172, 204-205, 218, 228, 236, 239, 241, 287-288
- file permission bits ... 24-25, 30-31, 92-93, 95, 98-99, 107-108, 111, 287-288
- file pointer ... 85, 160, 192, 247, 260, 282-283
- file prior ... 123-124, 222
- file record locking ... 35, 129, 247-249
- File Removal ... 100, 237
- file serial number ... 25, 37, 106, 202, 233, 288
- file size ... 34, 106, 126, 133, 198, 215, 234, 250, 264, 285, 288
- file space ... 25, 35, 98, 100, 125-126, 198, 213, 244
- File Status ... 108, 239
- file status flags ... 92, 124-125, 127-129, 131, 247
- file store requirements ... 267
- file system ... 25, 28, 31, 35-36, 42, 87, 92, 94, 97-101, 103-104, 106, 110, 112-113, 115, 117, 120, 137, 169, 172, 177, 189, 198, 201-207, 215-217, 222, 228, 233, 236, 238-242, 244, 246, 256-257, 261-266, 274, 287

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- file types ... 24, 26, 31, 37, 106, 129, 131-132, 160-161, 170, 172, 202, 233, 240-241, 249-250, 260, 263-265, 267, 277, 285, 288
- filename ... 23-24, 27, 31-32, 38, 50, 87, 193, 201, 203-207, 217-218, 233, 235, 276, 287
- filename portability ... 31, 38, 193, 204-205
- fileno ... 160, 192, 203
- Files and Directories ... 233
- find ... 212, 214, 240, 258, 263
- FIPS ... 180, 261
- first-in-first-out ... 24, 119, 248
- fixed size ... 169, 234, 245, 285
- flag ... 50, 54, 60, 66, 92, 119, 123-131, 135, 137, 140-141, 143, 146, 226, 236, 241-242, 246-247, 280, 283-284, 288, 293
- floating point values ... 211
- flock ... 130-131, 249
- flush ... 142, 147, 151, 219
- fmod ... 155
- fold ... 38, 205-207
- fopen ... 155, 161, 202, 260
- for example ... 31, 34-35, 42, 59, 83, 122, 125, 139, 146, 184-185, 195, 198, 200-202, 204, 213-215, 218, 223-224, 240-241, 247-250, 252, 257, 263, 266
- foreground ... 25, 136, 149, 153, 222, 253
- foreground process ... 25, 136, 149, 153, 222, 253
- foreground/background checks ... 222, 253
- Foreword ... 3, 182, 192, 194
- fork ... 27-28, 47-48, 52, 55, 61, 71, 73, 78, 83, 122, 131, 136, 216, 218, 247-248, 260, 277, 294
- form ... 3, 22, 37, 50, 66, 83, 156-157, 180, 185, 202, 206, 232, 237, 240, 247, 251, 256, 275, 292
- format ... 3-4, 6, 35, 37-38, 80, 87, 156-158, 169, 172-173, 190, 195, 200-201, 210, 233, 244, 251, 257, 260, 262-267, 285, 288
- Format of Directory Entries ... 87, 201, 233
- format parameters ... 244
- format-creating utility ... 169, 265-266, 287
- format-reading utility ... 169, 173, 265-266, 287-288
- former ... 189, 247, 263
- fpathconf ... 45, 116-117, 240, 274, 276
- fprintf ... 155, 250
- fputc ... 155
- fputs ... 155
- framing ... 142-143
- fread ... 155
- free ... 35, 126, 155, 193, 220, 235
- freopen ... 155
- frexp ... 155
- fscanf ... 155
- fseek ... 155-156
- fstat ... 30-31, 61, 106, 108-109, 239
- FTAM ... 179
- ftell ... 155
- Full Use ... 6, 45, 186, 189-190, 194, 239, 263, 267, 285, 291
- full-duplex mode ... 137
- function ... 4, 18, 20-22, 25, 27-29, 31-38, 42-104, 106, 108-117, 119-129, 131-133, 135-136, 139-141, 146-156, 158, 160-163, 166-168, 183-185, 190-195, 199-200, 204, 207-209, 211-212, 215-216, 218-222, 224-231, 233-241, 243-244, 246-248, 250-253, 255-261, 273-277, 281, 283-284, 287-288, 291-294
- function address ... 21, 27-28, 61, 200, 208, 235, 255, 275-276
- function argument ... 21, 34, 53-54, 61, 64-67, 69, 80, 90, 96, 103, 108, 110, 114, 121, 132, 147, 150, 152, 154, 162, 218-219, 226, 235, 239, 243, 251-252, 283, 292
- function call ... 22, 29, 32, 34, 44-45, 47-49, 51, 54-55, 61-62, 64, 66-71, 73-74, 77-79, 89, 95, 111, 120, 146, 154, 162-163, 166, 168, 183, 199, 207, 216, 218-220, 224, 226-229, 234, 238-239, 243, 250, 252-253, 258-259, 287, 293
- function descriptions ... 21, 33, 133, 156, 199, 208
- function fails ... 51, 63, 67-68, 77, 83, 90, 96, 100-101, 109, 131, 216, 239
- function reads ... 31, 35, 108, 122-123, 139, 146, 148, 151, 166, 168, 233, 239, 243, 246-247, 251, 253
- function returns ... 28, 32-34, 48, 51, 53-56, 61, 63, 65-66, 69-74, 77-79, 81-86, 88-89, 92, 94, 103, 106, 111, 115, 117, 120-122, 125, 133, 135, 146, 153, 160-161, 163, 167, 191, 207, 218, 220, 226-227, 229-230, 233-235, 243, 246, 248, 253, 259, 292-293
- function sets ... 31-32, 45, 48, 51, 54, 61, 63-67, 69, 71-72, 74-75, 78-79, 93-94, 111, 114, 119-121, 125, 133, 135, 141, 146-147, 149, 154, 158, 163, 184-185, 190, 193, 207-208,

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 220, 225-227, 230, 234-235, 246, 248, 258, 260, 275-276, 293
- functions return ... 28, 32-34, 48, 51, 53-56, 61, 63, 65-66, 69-74, 77-79, 81-86, 88-89, 92, 94, 103, 106, 111, 115, 117, 120-122, 125, 133, 135, 146, 153, 160-161, 163, 167, 191, 207, 218, 220, 226-227, 229-230, 233-235, 243, 246, 248, 253, 259, 292-293
- future direction ... 182, 237, 276, 282
- fwrite ... 155, 250
- F_DUPFD ... 120, 127, 129, 131-132
- F_GETFD ... 127, 129, 131
- F_GETFL ... 127, 129, 131
- F_GETLK ... 127, 130-132
- F_OK ... 43, 110
- F_RDLCK ... 127, 130, 132
- F_SETFD ... 127, 129, 131, 247
- F_SETFL ... 127, 129, 131, 247
- F_SETLK ... 127, 130-132, 248
- F_SETLKW ... 127, 130-132, 248
- F_UNLCK ... 127, 130
- F_WRLCK ... 127, 130, 132
- general ... 5, 22-23, 29-30, 33, 92, 135, 137, 149, 155, 163, 186-187, 191, 193-194, 196, 200, 202, 204, 207, 224, 226, 235-236, 248, 251-252, 255, 261, 263, 265, 277
- general concepts ... 30, 200, 204, 207, 263
- General File Creation ... 92, 236
- general terminal function ... 149, 251, 255
- general terminal interface ... 23, 29, 135, 149, 251-252, 255
- General Terms ... 22, 193, 200, 277
- General Utilities ... 155
- Generate Terminal Pathname ... 84, 231
- generates ... 32-33, 36, 46, 59-60, 63, 66, 77, 84, 140, 142, 148, 199, 216, 221-224, 231, 255, 283
- generation ... 60, 143, 146, 216, 223
- get ... 73, 75-76, 78, 81, 85, 108, 116, 127, 129-130, 149, 153, 155, 214, 221, 227-230, 233, 239, 241, 246, 248, 250, 254-255, 284
- getcwd ... 91, 235-236
- getegid ... 61, 73-74, 277
- getenv ... 83, 155, 193
- geteuid ... 61, 73-74, 277
- getgid ... 61, 73-74, 277
- getgrent ... 166, 168, 274
- getgrgid ... 166
- getgrid ... 274
- getgrnam ... 166, 274
- getgroups ... 61, 75-76, 190, 236-237, 274
- gethostid ... 230
- gethostname ... 230
- getlogin ... 76-77, 167-168, 274
- getpgrp ... 61, 78, 229
- getpgp2 ... 61, 78, 229
- getpid ... 61, 64, 73, 78
- getppid ... 61, 73
- getpwent ... 78, 167-168, 274
- getpwnam ... 77, 167-168, 274
- getpwuid ... 77-78, 167-168, 274
- getuid ... 61, 73-75, 77, 209, 277
- getwd ... 235
- gid ... 74-75, 166, 172, 217, 228, 286-289
- GKS ... 179
- global ... 3, 176, 195, 268
- Global Externals ... 195
- GMT ... 202
- gmtime ... 156
- gname ... 286-287, 289
- granularity ... 138
- graphics ... 17, 177, 179, 204
- Graphics Standards ... 179
- greater ... 33, 42, 49, 51, 63, 77, 79, 84, 91, 122-123, 125, 129, 132, 138, 154, 229, 231, 243, 246, 252
- Greenwich ... 156-157, 202, 256
- group ... 3-5, 7-8, 22-31, 37, 41, 44-48, 50-51, 56, 58, 62-63, 73-76, 78-79, 92, 98-99, 106-107, 110-113, 135-136, 140-141, 148-149, 153-154, 165-167, 169-170, 172, 176-192, 194, 196, 198, 202, 204, 206-207, 209-211, 215-216, 218-223, 226-230, 232-237, 239-240, 242-244, 246, 250-253, 255, 260-263, 274-275, 277-278, 280-281, 283, 285, 287-289, 291-292
- group database ... 165-166, 261-262, 274
- group database access ... 166, 262, 274
- group databases ... 165-166, 261-262, 274
- group file ... 23-26, 29, 31, 45-46, 50, 92, 106-107, 110-113, 136, 154, 202, 207, 215, 218, 228, 236, 240, 244, 246, 261-262, 280, 285, 287-289
- group ID ... 23-25, 27-30, 37, 41, 44-47, 50-51, 56, 63, 73-76, 78-79, 92, 98-99, 106-107, 110-113, 153-154, 165-167, 189, 204, 209, 218, 221, 227-229, 240, 255, 278, 280-281, 288-289, 292
- Group ID number ... 167
- group name ... 5, 165-166, 196, 207
- grouplist ... 75-76
- groups match ... 24, 47, 56, 75, 79, 111, 218, 230, 255, 262

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- grp.h ... 166, 173, 209, 289
- gr_gid ... 166
- gr_mem ... 166
- gr_name ... 166
- guidance ... 5, 155, 180, 183, 188, 197, 268
- guide ... 5, 155, 180, 183, 188, 191, 197, 268
- handling ... 17, 57, 66, 70, 125, 135, 139, 155-156, 158, 163, 177, 191, 201, 203, 206, 208, 222, 224, 233, 236, 244, 246, 248, 257-258, 261, 264-265, 274-275
- Hang up ... 144-145
- hard link ... 263-264
- hardware ... 29, 35, 52, 58-59, 80, 143-144, 146, 182, 184-185, 193, 198, 211, 244, 247, 250, 252
- hardware control ... 144, 146, 252
- header ... 6, 20, 22, 33-34, 37-39, 43-45, 53, 57, 66-67, 80, 87-88, 106, 109, 114, 127, 129-130, 133, 142, 169-171, 193, 195, 199, 209, 211, 232-233, 238, 264, 275, 285-289, 291
- header block ... 171, 285-288
- header file ... 43, 45, 106, 169-171, 195, 233, 238, 264, 275, 285, 287
- Header File and Data Structure ... 106, 238
- Header Files ... 195
- hertz ... 232
- heterogenous file system ... 244
- hierarchy ... 31, 177, 202, 204-205, 236, 263, 277
- high level approach ... 251
- highlight ... 139
- historical implementation ... 183, 185, 188, 192, 201-202, 205-206, 208, 210, 213, 217-218, 220, 230, 233-234, 236, 238-239, 242, 246-247, 249-251, 253, 257, 269
- historical reasons ... 238, 243, 257
- historical term ... 191, 231, 251
- historically ... 185, 202-203, 206, 209, 219, 233, 257
- history ... 177, 257, 269
- holding ... 98, 130-131, 249, 288-289
- HOME ... 37
- home directory ... 37, 167
- however ... 4, 34, 45, 60, 136, 138-139, 146, 149, 153, 163, 172, 183-187, 193, 196, 200-201, 205, 213, 216, 218-219, 221-223, 226, 232, 236, 240-241, 243-244, 248, 251, 253, 255, 259, 261, 264, 266, 287
- HUPCL ... 141, 144, 146
- i-node ... 233, 264
- i-node number size ... 264
- i.e. ... 38, 121, 129, 138-139, 158-159, 170, 202, 205, 207, 218, 221, 225, 258-259, 263
- I/O ... 17, 24-25, 35, 87, 92, 119, 139, 142, 169, 177, 241, 244, 249, 276, 282-283, 285
- I/O operation ... 24, 35, 249
- ICANON ... 140, 147
- ICRNL ... 142-143
- ID ... 23-30, 36-37, 41-42, 44-48, 50-51, 54-56, 62-63, 73-79, 82, 92, 98-99, 106-107, 110-115, 130-131, 153-154, 165-167, 189, 194, 201, 204, 209, 215, 217-219, 221, 225, 227-230, 239-240, 255, 278-281, 288-289, 292-293
- IDs ... 225, 255
- IEEE ... 3-5, 7, 20-21, 27, 45, 62, 155-156, 158, 172, 175-176, 178, 181-182, 185-187, 194-196, 200, 237, 262, 268, 273
- IEEE 802.2 ... 178
- IEEE 802.3 ... 178
- IEEE 802.4 ... 178
- IFS ... 37, 277
- IGNBRK ... 142
- IGNCR ... 142-143
- ignoring ... 50, 57-60, 67-69, 136, 141-144, 146, 148-149, 193, 202, 205, 220, 222-224, 240, 265-266, 287-289
- IGNPAR ... 142-143
- illegal file ... 250, 283
- image ... 33, 35, 49-52, 217
- implementation ... 4, 19-22, 24-34, 38-39, 41-45, 47-48, 50, 52-61, 63, 66, 69, 76-81, 83-87, 92-93, 97-99, 101-103, 106, 108-113, 117, 121, 123-126, 129, 131, 135-138, 140-144, 146, 148-149, 151, 153-154, 156, 158-159, 163, 165, 168-171, 173, 176, 183-185, 188, 191-193, 196-198, 200-231, 233-251, 253, 257-258, 260-266, 269-270, 277, 280, 287-288, 291-293
- implementation characteristics ... 28, 48, 198, 277
- implementation conformance ... 4, 19-21, 25, 27, 39, 41, 43-44, 176, 184-185, 192-193, 197, 201, 210, 212-214, 217, 221-223, 226, 241-242, 244
- implementation conforming ... 4, 19-21, 25, 27, 39, 41, 43-44, 176, 184-185, 192-193, 197, 201, 210, 212-214, 217, 221-223, 226, 241-242, 244
- implementation connotations ... 261

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- implementation defined ... 19-22, 26, 28-30, 33-34, 43-45, 48, 53-55, 58-60, 63, 66, 69, 76, 78, 80-81, 83-85, 87, 109, 111, 113, 117, 126, 135, 144, 151, 156, 158-159, 163, 169-171, 173, 176, 183, 192, 196-197, 201-203, 205, 207-209, 217, 220, 222, 224-225, 229, 231, 235, 237, 239, 250, 257, 262-263, 265, 277, 292
- implementation dependent ... 223, 250
- implementation details ... 183, 204
- implementation independent ... 233, 263
- implementation permitting ... 21-22, 38, 201, 207, 212, 216, 224, 227-228, 230-231, 236, 238, 240, 244, 246, 260, 266, 288
- implementation recommendation ... 19
- implementation-dependent ... 210, 217, 219, 233, 260, 279
- implementation-specific ... 217
- implementor ... 3, 191, 196, 200, 205, 219-220, 226, 230, 240, 252, 257, 260, 263, 267
- importance ... 18, 273
- important ... 139, 192, 197, 207, 219, 249, 269
- inadequacies ... 239
- include file ... 24, 42, 106, 192, 200, 203, 217, 236, 238, 241, 244, 247, 256-257, 267
- includes ... 4, 7-8, 17, 21-22, 24, 27, 29, 32, 41-43, 50, 53, 56-57, 59, 62-69, 73-76, 80-82, 84-85, 87-88, 92, 95, 97, 99, 106, 108, 110-112, 114, 127-129, 133, 137, 142, 146, 149, 151, 153-154, 156-157, 160-163, 166-167, 171, 175-177, 181-189, 191-192, 194-196, 199-200, 203-207, 211, 213, 216-217, 221-222, 224-226, 229-231, 234-239, 241, 243-244, 247, 250-252, 256-257, 260-261, 263-264, 267, 273-276, 283, 287
- inclusion ... 188, 212, 231, 238, 262, 281
- incomplete pathname ... 38
- increasing ... 43, 187, 214
- individual timers ... 186, 231
- Industry Open Systems Publications ... 180
- infinite hangs ... 227
- influential paper ... 269
- information ... 3-5, 18, 20, 25, 30, 37-38, 53, 80, 82, 108-109, 127, 130, 138, 149, 153, 165, 169-171, 176, 180-181, 187-188, 190, 196, 198-199, 201, 203, 211, 215, 217, 225, 233, 251-252, 256-257, 261, 264, 270, 287-288, 291-293
- inheritance by init ... 222
- inherits ... 48, 51, 131, 136, 219, 222, 247-248, 260, 277-278
- init ... 219, 222, 225
- initial user program ... 165
- initial values ... 143-144, 146, 148, 160
- initial working directory ... 37, 165, 207
- initial working directory field ... 165
- initialization ... 48, 60, 64-65, 98-99, 162, 197, 207, 216, 222, 225, 233, 288
- INLCR ... 142-143
- inline ... 200
- ino_t ... 37, 106
- INPCK ... 142-143
- Input and Output ... 87, 119, 122, 241, 243
- Input and Output Primitives ... 87, 119, 241
- input and/or output ... 140
- input character ... 135-141, 143, 147, 213, 253-254
- input control value ... 143, 284
- input modes ... 42, 137-138, 141-142, 147, 253-254, 284
- input parity checking ... 142-143
- input processing ... 23, 42, 135-138, 141-142, 147, 253-254
- Input Processing and Reading Characters ... 136-137, 253
- input queue ... 41, 137, 139, 142-143, 147, 213, 253
- Input/Output ... 155
- instance ... 34, 41-44, 163, 203, 213-215, 232
- Institutional Representatives ... 186-187
- int ... 47, 49, 53, 55, 62, 64-69, 71, 73-75, 78-80, 85, 88, 90-92, 96-97, 99-100, 102-103, 108, 110-112, 114, 116, 119-120, 122, 124, 128-130, 133, 145, 149, 151, 153-154, 158, 160-163, 199, 209, 212, 220, 225, 227, 234, 243, 245, 258, 275, 278, 291
- integer ... 24-25, 27, 30, 45, 94, 119, 123, 126, 129, 158, 160, 222, 230-231, 243, 288
- integer values ... 25, 30, 45, 119, 123, 158, 222, 230-231, 288
- integral ... 37, 57, 209
- integral types ... 37, 57, 209
- intent ... 4, 221, 239, 265
- intention ... 184, 214, 216, 250, 261
- inter-process communication ... 189
- interactive attention signal ... 58
- interactive stop signal ... 59
- interactive termination signal ... 58

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- intercharacter ... 138-139
- interface ... 3-4, 17, 20, 22-23, 25, 29, 54,
 - 120, 135-136, 141, 144-145, 149,
 - 163, 175-178, 181-185, 188-192,
 - 195-196, 200, 202-206, 208, 210,
 - 219-221, 228, 230, 236, 240-241,
 - 243, 250-253, 255, 257, 261-263,
 - 273, 275
- interface characteristics ... 135, 253, 273
- interim ... 21, 155, 256
- internal construction techniques ... 18
- internal static area ... 84
- international applications ... 182, 232, 258
- interpretation ... 27, 32, 165, 171, 192, 200, 266
- interpreter ... 25, 29, 38, 135, 223
- interrupted call ... 34, 62, 208, 220
- interrupted operation ... 243
- interval ... 23, 60, 216, 223
- interval invisible ... 216, 223
- INTR ... 140-141, 147-148, 254
- introducing function ... 190, 221, 227, 238, 252
- introduction ... 182, 195
- INT_MAX ... 40, 123, 125, 243
- INT_MIN ... 40
- ioctl ... 251
- ioctl ... 208, 250-252, 255, 274, 276, 284
- isalnum ... 155
- isalpha ... 155
- isascii ... 193
- isatty ... 85, 192, 208
- isctrl ... 155
- ISCTG ... 264
- isdigit ... 155
- isgraph ... 155
- ISIG ... 140, 147
- ISLKN ... 265
- islower ... 155
- ISO ... 20, 29, 175, 178-179, 196, 206, 259
- ISO member body ... 20
- isprint ... 155
- ispunct ... 155
- ISSOCK ... 265
- isspace ... 155
- ISTRIP ... 142-143
- ISUID ... 263
- isupper ... 155
- isxdigit ... 155
- IXOFF ... 141-143
- LXON ... 141-143
- jcgetpgrp ... 61
- jcsetpgrp ... 25, 28-29, 61, 79, 153, 229-230, 255, 275
- job ... 22, 25-26, 28-29, 44, 47, 54, 56, 59-60, 63, 66, 79, 135-136, 140-141, 147-149, 153-154, 179, 189, 218-226, 229-230, 232, 250, 253, 255, 275, 278-279, 283, 291-293
- job access control ... 22, 25, 59, 136, 221-222, 253
- Job Control Option ... 25-26, 28-29, 47, 54, 56, 59-60, 63, 66, 135-136, 140-141, 147-149, 275, 283, 291-293
- job control process group leaders ... 25-26, 28-29, 79, 219-220
- job control shell ... 219-220, 222-225, 229-230, 253, 255
- job control signals ... 26, 59-60, 66, 140, 148, 219, 222-225, 253, 279, 283, 293
- job process group leader ... 25-26, 28-29, 79, 219-220
- Julian ... 157, 256-257
- kernel ... 179, 202-203, 206, 270
- Kernighan ... 192, 256
- KILL ... 138, 140-141, 147-148, 254
- kill ... 34, 44, 48, 57, 59, 62-63, 67, 71, 73, 78, 138, 140-141, 147-148, 191, 194, 216, 218-219, 222-225, 252, 254-255, 279
- Korn ... 210
- LANG ... 37, 159
- Language Binding ... 21, 199
- Language Conformance ... 21, 199
- Language Standards ... 176, 178
- languages ... 3, 5, 17, 21, 34, 36, 39-40, 49, 52, 56, 58, 62, 108, 155-156, 158, 160-163, 176-179, 184, 188-196, 199, 205-206, 211, 219, 222, 226-227, 234, 237, 239, 244, 254, 256-257, 259-261, 268, 271, 274
- large ... 34, 36, 76, 186, 209, 212-214, 221, 232-233, 235, 244, 250, 252, 261-262, 264
- last call ... 228
- latitude ... 208
- latter ... 186, 191, 193, 195-196, 198, 203, 208, 219, 226, 230, 233, 236-237, 247, 252-253, 261, 266
- latter function ... 191, 219, 237, 252
- latter term ... 191, 261
- latter type ... 191, 230
- LC_ALL ... 159, 258-259
- LC_COLLATE ... 37, 158-159, 258-259, 277
- LC_CTYPE ... 37, 158-159, 258-259, 277
- LC_NUMERIC ... 38, 158-159, 258-259, 277
- LC_TIME ... 38, 158-159, 258-259, 277

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- ldexp ... 155
- leap seconds ... 24, 202
- legal ... 250
- lengths ... 41-42, 51, 76, 90-91, 93-94, 97-98, 100-102, 104, 109-110, 112-113, 115, 139, 170-171, 204, 213, 230, 234-235, 264, 286
- library functions ... 21-22, 183, 191, 199, 218, 261
- library routine ... 203-204, 208, 218, 223, 261
- Library Routines ... 274
- limit ... 20-21, 33, 35-36, 38-40, 48, 51, 86, 112, 116, 120, 125, 132, 137-138, 142, 187, 193, 195-198, 202, 209, 211, 213-217, 220, 230, 232, 240-241, 246, 248, 251, 262-263, 267, 278, 287, 289
- limitations ... 32, 39, 41, 43, 240, 256, 267
- limits.h ... 20-21, 39, 41-43, 86, 116-117, 193, 198, 211-215, 232, 241
- line ... 6, 42, 122, 138, 140, 144, 146-147, 151, 192, 202, 213, 249, 251-252, 254-255, 283
- line control functions ... 151, 255
- line numbers ... 42, 122, 138, 213
- line speeds ... 251
- link ... 23, 26, 34, 36-37, 43, 46, 62, 96-98, 100-106, 108, 170, 173, 178, 201, 205, 236, 238, 262-265, 267, 280, 286-288
- link count ... 26, 34, 37, 43, 96, 98, 100-101
- Link Layer Control ... 178
- linkname ... 267, 286-288
- LINK_MAX ... 34, 43, 96, 98, 116
- link_t ... 238
- list ... 7-8, 20-21, 32-33, 37-39, 50-51, 62, 83, 86, 92, 96, 101, 106, 108, 116, 165, 175, 178, 180, 185, 188-189, 191-192, 198, 200, 216-217, 224-226, 236, 239-240, 256, 273, 277, 279, 282
- LNKTYPE ... 286, 288
- local control value ... 148
- local file system ... 169, 287
- local modes ... 136-137, 142, 147, 254, 284
- locale ... 37-38, 136-137, 142, 147-148, 156-159, 169, 254, 256-259, 284, 287-288
- localtime ... 156, 256
- lock ... 33, 35, 48, 50, 121, 127, 129-132, 187, 247-249
- locked region ... 130-132, 247-248
- lockf ... 33, 35, 48, 50, 121, 127, 129-132, 187, 247-249
- locking process ... 33, 50, 121, 130-132, 187, 248-249
- locking requests ... 130-132, 248
- lockout ... 248
- LOCK_MAX ... 215
- log10 ... 155
- logical device ... 203
- login name ... 38, 77, 165, 167, 236, 274
- login shell ... 215, 219, 222
- LOGNAME ... 38, 211, 277
- longjmp ... 155, 162, 194, 224, 227, 244, 260
- LONG_MAX ... 40
- LONG_MIN ... 40
- look ... 215, 227, 255
- loops ... 235-236
- lower ... 146, 204
- lowercase ... 38, 156, 193, 195, 205-206
- lread ... 209, 243-244
- lseek ... 24, 36, 44, 62, 95, 124, 126, 133, 156, 209, 212, 215, 249-250, 283
- lvalue ... 234
- lwrite ... 243-244.
- L_ctermid ... 84, 231
- L_cuserid ... 77, 229
- l_len ... 130-131
- l_pid ... 130-131
- l_start ... 130-131, 249
- l_type ... 127, 130, 132
- l_whence ... 130-131
- machine ... 80, 146, 206, 214-215
- macro ... 22, 106-107, 162, 199, 260, 284
- magic ... 170, 286-287, 289
- magic bytes ... 170
- magic field ... 287, 289
- magnitude ... 39, 41, 43
- mail ... 6, 38, 185, 277
- main ... 4, 29, 49, 52, 56, 60, 193-194, 208, 210, 217, 219, 247, 278
- main body ... 194
- main memory ... 208
- major ... 17, 184-185, 189, 195, 198, 219-220, 244, 248, 250
- major difference ... 198, 220
- major feature ... 198, 248
- make directory ... 97, 218, 236-237
- malloc ... 155, 193, 235
- mandatory ... 187, 205, 248-249
- mandatory locking ... 187, 248-249
- mandatory locks ... 187, 248-249
- manipulate signal sets ... 64, 225
- manipulating ... 39, 57, 64, 225, 276
- manipulation ... 60, 120, 177, 243
- manner ... 33, 67, 114, 137, 156, 158-159, 170-171, 173, 186, 224-225, 240, 242, 252, 265, 287, 289

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- manual ... 188-189, 194, 269-270
- many ... 26, 34-35, 89, 94, 125, 138, 177, 183-184, 186-189, 192-194, 198, 200, 205-206, 208-211, 214-216, 218, 220, 226-227, 230, 232-233, 236, 238, 240, 243, 248-249, 252-253, 257, 260-261, 265, 274
- margin ... 6
- mask ... 51, 53, 60, 66-69, 93, 95-96, 98-99, 107, 128-129, 142, 144, 147, 162-163, 220, 223, 226, 236, 260, 275, 292
- materials ... 3, 176, 182, 185, 194, 215
- maximum ... 34, 41-43, 126, 171, 209, 212, 235, 241, 244-246, 266, 287, 289
- maximum pathname length ... 42, 235
- maximum portability ... 287
- MAX_CANON ... 42, 116, 138
- MAX_CHAR ... 213
- MAX_INPUT ... 41, 116, 137-138, 143, 213
- may ... 3-4, 19-20, 22, 24-25, 27-29, 31-35, 37-39, 42-45, 53-55, 57, 59, 61, 63, 66, 69-70, 72, 77-78, 80-85, 87-89, 92, 103, 106, 109-113, 122, 125, 129, 131, 135-141, 146, 148-149, 153, 157, 159, 161, 163, 165, 167-169, 172-173, 176-177, 181, 184, 188, 190-192, 196-200, 204-216, 219, 222-223, 225-228, 231-232, 234, 236, 238-245, 247-250, 253-257, 259-261, 264-265, 267-270, 273, 275, 277-279, 285, 287-289, 292, 294
- meaning ... 19, 22, 24, 37, 59, 137-138, 141, 157, 163, 177, 191-192, 196, 201-202, 204, 211, 232-233, 238, 246, 254, 257, 260, 267, 279, 288
- mechanism ... 29-31, 138, 140, 169, 177, 189, 193, 200, 202, 204, 220-222, 224-226, 230, 233, 247-250, 263-264, 267
- medium ... 125, 169, 263, 265-266, 285, 287
- meeting ... 20, 194, 198, 212, 240, 242, 261-262, 266-267
- members ... 7-8, 20, 24-25, 27, 29, 48, 50, 65-66, 80, 82, 87, 106, 114, 130, 136, 142, 148, 166-167, 182, 186-187, 214, 216, 221, 230, 240, 270, 280
- memory ... 22, 35, 42, 52, 58, 208, 213, 217, 249-250, 274
- memory management ... 35, 52
- metafile ... 179
- method ... 86-87, 116, 202, 205, 216, 222, 233-235, 244, 251-253, 256-257, 262
- might ... 183, 201-202, 204, 216, 232, 240, 263
- MIN ... 138-139, 147-148
- minimal changes to existing application code ... 185, 206, 225
- Minimal Changes to Historical Implementations ... 185, 202
- Minimal Directory Tree Structure ... 275
- minimum ... 41-43, 138-139, 177, 212, 232, 240, 263-264, 267
- minimum number ... 138-139, 212, 264
- minimum requirements ... 267
- minimum value ... 41-43, 212, 232, 263-264, 267
- mkdir ... 62, 95-98, 103, 108, 112, 173, 190, 208, 228, 237-238, 277
- mkfifo ... 62, 95-96, 99, 108, 112, 202, 228, 237, 274
- mknod ... 202, 237, 274
- mode ... 22, 24-26, 34, 42, 50-51, 58, 62, 92-93, 95-99, 106-107, 111-112, 128-129, 131, 136-142, 144, 146-148, 161, 169, 178, 190, 192, 194, 197, 229, 239, 249-254, 260, 263-264, 267, 282-284, 286-288
- mode field ... 169, 287
- modem ... 22, 24-26, 34, 42, 50-51, 58, 62, 92-93, 95-99, 106-107, 111-112, 128-129, 131, 136-142, 144, 146-148, 161, 169, 192, 239, 249-254, 260, 263, 282-284, 286-288
- modem access ... 22, 26, 92, 111, 128-129, 131, 249
- modem connection ... 146
- modem control lines ... 146
- modem disconnect ... 58, 141, 146, 254
- modem line control ... 146
- modem lines ... 138, 146, 192, 251
- modem status lines ... 146
- mode_t ... 37, 92, 95, 97, 99, 106, 111, 209, 236-238, 277
- modf ... 155
- modification time ... 106, 114, 171, 240, 288
- modifying ... 17, 36, 66, 76, 251, 292
- modtime ... 114, 240
- most characters ... 138, 230
- most functions ... 32, 47, 185, 191, 208, 230, 233, 239-240, 246
- mount ... 202-204, 240, 274
- mount point ... 203-204
- mounted file system ... 202-203, 240
- mtime ... 286-288
- mtime field ... 287-288
- mtimer ... 286-288

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- much data ... 137
- multi-byte ... 141
- Multi-volume archives ... 265
- Multics ... 269
- Multiple Groups ... 275
- Multiple Volumes ... 173, 265
- Mumps ... 178
- must ... 25, 30, 45, 62, 106, 108, 114, 154, 161, 169, 186-187, 194, 198, 200, 204, 207, 209-210, 212, 215, 219, 222-223, 228-229, 231-234, 236, 239-240, 248-249, 253, 259, 265-266, 277, 281
- mutability ... 211
- name ... 5, 7, 20, 23-25, 33-34, 37-39, 41-43, 57, 66, 76-77, 80, 82-83, 85-87, 92, 96, 98, 100, 102-104, 106, 110, 114, 116-117, 130, 136, 142, 144-145, 147-148, 158, 165-172, 183, 188-189, 194-196, 200-201, 203, 205-208, 210, 218, 220-223, 229-231, 233-234, 236, 238, 241-242, 256-260, 264-265, 267, 274, 283, 286-288
- name field ... 170, 267, 287-288
- name path ... 96, 98, 100, 102, 104, 110, 116-117, 171, 203, 210
- name.h ... 23, 38-39, 77, 189, 211, 234
- named directory stream ... 88, 235
- names starting ... 200
- NAME_MAX ... 24, 32, 35, 42, 46, 51, 87, 90, 94, 97-98, 100-102, 104, 109-110, 112-113, 115-116, 216, 240-241
- NAMSIZ ... 267, 287
- NBS ... 180, 261
- nbyte ... 122-125, 243, 245-246
- NCCS ... 142, 148
- NDL ... 179
- need ... 18, 22, 70, 139, 142-143, 172, 178, 197, 205-206, 209, 214-217, 226, 228, 232-234, 239, 241, 244, 251-252, 257, 259-261
- negative value ... 63, 131, 227, 233
- network ... 17, 80, 135, 146, 177-179, 181, 184, 190, 205, 207, 209, 238, 244, 248, 251-252
- network connection ... 135, 146
- networked systems ... 178, 181, 190, 207, 244, 251-252
- networked transfers ... 244
- networking ... 17, 80, 135, 146, 177-179, 181, 184, 190, 205, 207, 209, 238, 244, 248, 251-252
- networking standards ... 178, 190, 207, 252
- NGROUPS_MAX ... 29, 41, 76, 86, 275
- NL ... 140, 142-143, 147
- nlink_t ... 37, 106, 209
- no-op ... 225
- nodename ... 80, 230
- NOFLSH ... 147
- Non-canonical mode ... 254
- non-canonical mode input processing ... 137-138, 147, 254
- non-local jumps ... 155, 162, 194, 221, 260, 274
- non-negative integer ... 24, 94
- non-negative value ... 81
- non-null ... 217, 287
- non-null characters ... 287
- non-reentrant function calls ... 224
- non-standard signals ... 223-224
- non-variable ... 225
- non-zero ... 22, 25, 72, 107, 125, 136, 162, 234
- normal ... 34, 52, 61, 125, 146-147, 195, 200, 248, 263
- normal circumstances ... 146
- normal completion ... 125
- normal flush ... 147
- normal return ... 34, 52, 61, 125
- normal termination ... 52
- normal usage ... 195, 248
- normally ... 22, 33, 36, 66, 135, 146, 195, 202, 220, 224, 226
- note .. 3, 6-8, 45, 138-139, 143, 181, 184-185, 188, 191-195, 200-202, 221, 223-225, 230, 233, 239-241, 244, 247-248, 253, 255-256, 260, 265, 267-268, 279, 289
- notes sections ... 3
- notification ... 221, 223
- notion ... 204
- NUL ... 156
- NULL ... 21, 39, 46, 49-50, 66-68, 77, 81, 83-85, 88-89, 91, 114-115, 161, 167-168, 234-235, 258-259
- null byte ... 24, 27, 39, 41-42, 50, 171, 206, 229, 231, 264
- null character ... 24, 27, 50, 77, 84, 87, 91, 206, 233, 287
- null define ... 286
- null password ... 41
- null pathname ... 32, 42, 44, 264
- null pointer ... 21, 39, 49-50, 77, 81, 83-85, 88-89, 91, 114, 158-159, 161, 167-168, 259
- null signal ... 57, 62-63, 66-68
- null string ... 32, 41-42, 50, 77, 83-84, 158, 233, 259, 287

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- null-terminated ... 50, 80, 85, 87, 166, 287
- null-terminated character array ... 50, 80, 287
- null-terminated character string ... 50, 287
- Null-terminated filename ... 87
- null-terminated pathname ... 85
- null-terminated string ... 50, 287
- number ... 4-6, 20, 22-23, 25-26, 32-35, 37-38, 41-43, 48, 50-51, 53-54, 61, 64-65, 67, 70, 72, 76, 87, 94, 96, 106, 120-126, 129, 131-132, 137-139, 143, 148, 157, 165, 167, 170, 180, 183, 187-188, 191, 193, 195, 197, 201-203, 207-208, 212-214, 217, 220, 224-225, 227, 233-234, 236, 243-248, 256, 262, 264, 273, 275-276, 278, 287-289, 292-293
- numeric editing ... 38, 258
- numeric value ... 112, 230
- numerical group ID ... 165-166
- numerical limits ... 39, 193, 211
- numerical user ID ... 165
- numerous ... 189, 250, 267
- numerous enhancements ... 267
- object ... 17-18, 23-24, 27, 57, 64, 87, 89, 146, 149, 151, 166-167, 184, 200, 204, 210, 222, 228, 232, 235, 242, 244, 249
- object compatibility ... 184, 210, 242
- object file ... 23-24, 242, 249
- occur ... 29, 32-33, 35, 51-53, 56-57, 60, 70, 72, 91, 98, 102, 104, 131, 138, 140, 146, 149, 156-157, 171, 189, 193, 198, 207, 219, 226-227, 234, 236, 248, 255, 266, 282, 288, 292
- OCRNL ... 284
- octal ... 24, 53-54, 170, 264, 287-288, 292-293
- octal value ... 24, 53-54, 264, 287-288, 292-293
- odd parity ... 144, 146
- offsets ... 24, 26, 44, 92, 122, 124-125, 130-131, 133-134, 156-158, 244, 247, 249-250, 256-257, 283, 286
- off_t ... 37, 106, 130, 133, 209, 212, 238, 244, 249, 283
- OFILL ... 284
- offlag ... 92, 94, 128, 236, 247, 280
- OLCUC ... 284
- older function ... 235
- one-line tag ... 208
- ones ... 4, 20-21, 24-27, 29-33, 37, 42, 45, 47, 49-53, 62-63, 65-68, 70, 83, 92, 95, 100, 104, 111-112, 119, 122, 131, 137-140, 146-148, 154, 157-159, 161, 173, 175, 184-185, 191-195, 197-198, 200-203, 206, 213, 215-217, 220, 223, 226-227, 229-231, 235-236, 238-245, 248-249, 252-255, 257-259, 261-263, 265-266, 271, 273-274, 276, 287, 293
- ongoing efforts ... 18
- ONLCR ... 284
- ONLRET ... 284
- ONOCR ... 284
- open ... 5, 24, 26, 30-31, 33-35, 42, 47, 50, 55, 62, 88-90, 92-96, 100, 102, 108-109, 111, 116, 119-124, 126-137, 143-144, 146, 148-149, 152, 160-161, 175, 178, 180, 184, 195, 198, 202-203, 212, 214-217, 219, 222, 226, 228, 231, 235-236, 238-240, 242-243, 247-249, 253, 260, 265, 277, 280, 287
- open file ... 24, 26, 31, 33-35, 42, 47, 50, 55, 89, 92-95, 100, 108, 111, 116, 119-120, 122, 124, 126, 128-133, 135-136, 146, 149, 152, 161, 198, 203, 212, 214-217, 219, 222, 231, 236, 239-240, 243, 247-249, 253, 260
- open file description ... 24, 26, 47, 50, 92, 119, 129, 133, 203, 243, 247-248
- open file descriptor ... 26, 33-34, 47, 50, 55, 89, 92, 108, 111, 116, 119-120, 122, 124, 126, 129-133, 135, 149, 161, 203, 222, 231, 239-240, 243, 260
- open instance ... 203, 215
- opendir ... 88-89, 190, 287
- opening ... 5, 24, 26, 30-31, 33-35, 42, 47, 50, 55, 62, 88-90, 92-96, 100, 102, 108-109, 111, 116, 119-124, 126-137, 143-144, 146, 148-149, 152, 160-161, 175, 178, 180, 184, 195, 198, 202-203, 212, 214-217, 219, 222, 226, 228, 231, 235-236, 238-240, 242-243, 247-249, 253, 260, 265, 277, 280, 287
- opening special files prior ... 222
- opening terminal device file ... 135, 253
- OPEN_MAX ... 34, 42, 86, 120-121, 132, 195, 214-215
- OPEN_MAX_CEIL ... 214-215
- operating ... 3-4, 7, 17, 25, 47, 64, 80, 87, 137-138, 175, 177, 181-184, 188, 191-192, 195-197, 201-203, 205-206, 230, 236, 252, 263, 274, 288
- operating environment ... 3, 17, 183, 191-192, 201
- operating system ... 3-4, 7, 17, 25, 47, 64, 80, 87, 137, 175, 177, 181-184, 188,

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 191-192, 195-197, 201-203, 205-206, 230, 252, 263, 274, 288
- Operating System documentation ... 3
- Operating System Primitives ... 274
- operation ... 3, 17, 24, 34-36, 58, 70, 87-88, 94, 103-104, 112, 124, 126-127, 130, 136, 139, 149, 184, 190, 201, 204, 208, 214, 221, 226, 228, 234-235, 239, 243-249, 265, 282, 285
- OPOST ... 144
- option ... 21-22, 25-26, 28-29, 44-45, 47, 53-56, 59-60, 63, 66, 79, 86, 92, 116, 135-136, 140-141, 147-149, 153-154, 183, 187, 190-191, 196-198, 202, 205-206, 217-219, 228, 232, 257, 262, 275, 279, 283-284, 291-294
- optional actions ... 44-45
- optional facilities ... 44-45
- optional features ... 19, 275
- optional_actions ... 149-150
- order ... 4, 32, 37, 53-55, 88, 159, 169, 175, 197, 206, 211-212, 214, 217, 224-226, 228, 230-231, 238, 242-244, 252, 260-261, 268, 270, 278, 292-293
- ordinary application ... 231, 233
- organization ... 3, 31, 194-196
- original state ... 230, 261
- OSI Model ... 178
- other function calls ... 64, 77, 168, 207, 218, 220
- other reason ... 200, 219, 236, 257
- output ... 27, 34-35, 38, 87, 119, 122, 135, 137-138, 140-149, 151-152, 160, 241, 243, 254, 265, 284, 288-289
- output baud rates ... 145-146
- output characters ... 137-138, 140-141, 143-144, 147, 254
- output control value ... 144
- output modes ... 137, 140-142, 144, 254, 284
- output primitives ... 87, 119, 241
- output processing ... 135, 137, 140-141, 144, 148, 254
- output queue ... 137, 147
- overflowing ... 58, 83, 143, 231
- overwriting ... 130, 235
- overwritten ... 77, 84-85, 88, 167-168
- owner ... 24-25, 31, 36, 46, 50, 77, 93, 98-99, 106-107, 111-115, 170, 172, 218, 240, 249, 263, 281, 287-289
- owner ID ... 24-25, 50, 98-99, 106-107, 111-115, 218, 240, 281, 289
- ownership ... 112, 169
- O_ACCMODE ... 128-129
- O_APPEND ... 92, 124, 128
- O_CREAT ... 92-95, 128, 212, 280
- O_EXCL ... 93-94, 128
- O_NDELAY ... 241-242, 246-247, 280, 282-283
- O_NONBLOCK ... 93-94, 119, 123-126, 128, 135, 137, 146, 185, 241-242, 245-247, 280, 282-283
- O_RDONLY ... 92-93, 128
- O_RDWR ... 92, 94, 128, 212
- O_TRUNC ... 93-95, 128, 212
- O_WRONLY ... 92-95, 128
- P1003 ... 5, 182, 262
- P1003.1 ... 7, 181, 184, 188, 190-194, 196, 211
- P1003.2 ... 5, 135, 169, 218
- P1003.3 ... 5, 198, 214-215
- P1003.4 ... 5, 178
- page ... 8-15, 237
- pair ... 39, 170, 214, 242
- papers ... 268-269
- parameter ... 136, 142, 149, 163, 191-192, 199, 212, 222, 225-226, 229, 238, 240, 244, 251, 254, 276, 279
- PARENB ... 144, 146
- parent ... 26-27, 29, 32, 46-48, 51-52, 54-55, 60, 63, 73, 82, 93-94, 98, 101-102, 104, 216, 218-219, 224, 227-228, 230-231, 236, 255, 277, 292
- parent directory ... 26, 32, 46, 93-94, 98, 101-102, 104, 228, 236, 277
- parent following ... 47, 216
- parent process ... 26-27, 29, 46-48, 51-52, 54-55, 60, 63, 73, 82, 216, 218-219, 224, 227-228, 230-231, 236, 255, 292
- parent process ID ... 26, 29, 46-48, 51, 54-55, 63, 73, 82, 219, 227-228, 255, 292
- parity ... 142-144, 146, 287
- parity error ... 143
- PARMRK ... 142-143
- PARODD ... 144, 146
- part ... 3, 17, 25, 30, 37-39, 101, 162, 175, 181, 189, 192, 194-195, 198, 207-208, 219, 225, 248, 252, 261, 263, 265, 287
- participating ... 5, 8, 178, 186
- particular ... 31, 63, 88, 146, 159, 163, 185-186, 191-192, 207, 212, 214, 239, 245, 257, 260-261, 288
- Pascal ... 178
- passwd ... 165, 167, 203, 210, 261, 274
- passwd database ... 261, 274
- passwd file ... 203, 261

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- passwd.h ... 209
- Password ... 261
- password ... 37-38, 41, 77, 165, 168, 202-203, 249, 261, 289
- password database ... 37-38, 77, 165, 168, 249, 261
- PASS_MAX ... 41, 86
- PATH ... 38, 49, 195, 210
- path name ... 96, 98, 100, 102, 104, 110, 116-117, 171, 203, 210
- path prefix ... 26, 32, 38, 49, 51-52, 94, 96-101, 104, 109-110, 112-113, 115, 287
- path1 ... 26, 32, 38, 49, 51-52, 90, 92, 94-102, 104, 108-117, 171, 195, 203, 210, 212, 287
- pathconf ... 32, 42, 45, 116-117, 216, 240-241, 274, 276
- pathname ... 23-24, 26-28, 30-32, 34-35, 38, 42, 44, 46, 49, 51-52, 84-85, 90-92, 94, 96-104, 108-116, 170, 201, 203, 205, 207, 214, 216, 231-233, 235, 240-241, 247, 263-264, 267, 277, 287
- pathname component ... 24, 27, 32, 35, 46, 51-52, 90-91, 94, 97-98, 100-102, 104, 109-110, 112-113, 115, 216, 240-241
- pathname resolution ... 23-24, 27-28, 30-32, 201, 205, 207, 233
- PATH_MAX ... 27, 34, 42, 51, 90, 94, 97-98, 100-102, 104, 109-110, 112-113, 115-116, 213, 233, 235, 264, 267, 276
- pause ... 55, 62, 70-72, 226-227, 244-245, 294
- pausing ... 55, 62, 70-72, 226-227, 244-245, 294
- pclose ... 218-219, 274
- pending ... 48, 51, 60-61, 63-64, 68-69, 139, 163, 216, 220-221, 223, 226, 279
- pending signals ... 48, 51, 60-61, 63-64, 68-69, 163, 216, 220, 223, 226, 279
- performs ... 3, 26, 31, 34, 36, 47, 62, 87, 103, 141, 143-144, 149, 189, 202, 212, 222, 228, 239, 244, 246, 248, 253, 262-263, 289
- perhaps ... 5, 202, 217-219, 245, 260
- permission ... 4, 23-26, 29-31, 33, 35, 37, 43, 51-52, 62-64, 89-91, 93-96, 98-104, 107-115, 136, 169-170, 172, 202, 204-205, 218, 225, 228, 236, 239, 241, 287-288
- permission bits ... 24-25, 30-31, 93, 95, 98-99, 107-108, 111, 287-288
- permission checking ... 110, 225, 239
- permits ... 21-22, 36, 38, 110, 114, 191, 200-201, 204, 206-208, 212, 216, 224-225, 227-228, 230-231, 236, 238, 240, 243-244, 246-247, 255, 260, 266, 288
- perror ... 155, 208
- pxsiz ... 267, 287
- pgrp ... 79
- pgrp_id ... 154
- PHIGS ... 179
- physical end ... 125
- physical I/O operations ... 285
- pid ... 44, 62-64, 218, 225, 279, 291-294
- PID_MAX ... 27, 41, 63, 79, 86, 154, 292
- pid_t ... 209
- pipe ... 24, 27, 36, 43, 58, 62, 100, 108-109, 119, 121-126, 134, 202-203, 220, 242, 244-246, 255, 260
- pipeline ... 24, 27, 36, 43, 58, 62, 100, 108-109, 119, 121-126, 134, 202-203, 220, 242, 244-246, 255, 260
- PIPE_BUF ... 43, 116, 125, 244-246
- PIPE_MAX ... 246
- pitfalls ... 251
- point ... 25, 31, 37, 49-50, 53, 55, 61, 64-69, 76-77, 80-85, 88-92, 94, 96-104, 108-116, 122-124, 130-133, 138-139, 145, 158, 163, 167-168, 185-186, 195, 198, 201, 203-204, 210-211, 216-217, 219, 222, 231-232, 241, 251, 258, 273, 291, 294
- pointer ... 21, 39, 49-50, 60-61, 66, 77, 81, 83-85, 87-89, 91, 109, 114, 130, 145, 158-161, 163, 166-168, 192, 200, 217, 222, 247, 252, 259-260, 282-283
- popen ... 218-219, 274
- port ... 141, 146, 180
- portability specifications ... 44-45, 183, 215-216, 252
- portable ... 3, 17, 27, 31, 38, 155, 172, 175, 178, 181, 183, 193, 197, 203, 206-207, 219, 221, 223-225, 232, 250-252, 260-262, 264, 287, 289
- portable application ... 38, 155, 183, 197, 207, 223-225, 232, 250-251, 261
- portable filename character set ... 27, 31, 193, 203, 206, 287
- portable library ... 223, 250, 261
- portable mechanism ... 221, 224-225, 250
- Portable Operating System Interface ... 3, 17, 175
- portable way ... 224
- portion ... 25, 123, 130, 132, 258, 265

- position ... 4, 24, 88-89, 95, 122, 124, 131, 156, 206, 235, 249-250, 265
- POSIX ... 3, 5, 182, 188, 190-198, 201-203, 205, 211, 220, 222, 226, 233, 237, 239, 241, 247, 250, 252, 256, 258-261, 263-265
- posix.h ... 215
- possibility ... 214, 249
- possible error numbers ... 33
- possible errors ... 3, 32-33, 232, 265
- possible requirement ... 185-186, 192, 198, 213, 244
- possibly ... 42, 199, 201, 204, 214, 252
- potential ... 131, 184, 193, 215, 218, 261
- potential security problem ... 218
- potentially ... 220, 240-241
- practice ... 19, 135, 181, 183, 207, 210, 264
- precise ... 143, 190, 230
- precision ... 202, 288
- precluding ... 201, 204, 219
- predecessor ... 31-32, 246
- prefix ... 26, 32, 38, 49, 51-52, 94, 96-101, 104, 109-110, 112-113, 115, 167, 267, 286-287
- prefix field ... 267, 287
- prepending asterisk ... 200
- preprocessing directives ... 211
- presence ... 25, 200, 232, 279
- present ... 3, 20, 44-45, 50, 83, 139, 156-157, 188-190, 192-193, 197, 209-210, 216, 219, 229-230, 233, 235, 238, 240, 248, 256, 269, 275
- present form ... 3
- present standard ... 3, 20, 45, 157, 188-190, 192-193, 197, 209, 216, 219, 230, 233, 235, 238, 269
- presents problems ... 229, 248, 256
- prevent errors ... 32, 224
- previous draft ... 247
- previous lock type ... 131, 248
- previous version ... 234
- primitive system data types ... 37, 209
- primitives ... 37, 47, 64, 87, 119, 176-177, 209, 216, 219, 241, 274
- principal ancestor ... 188, 269
- printf ... 7-8, 155, 177, 194, 208, 223, 226, 270
- prior ... 53, 60, 64, 69, 72, 92, 123-124, 204, 216-217, 222, 233
- privileged operation ... 228
- privileges ... 22, 27, 30, 36, 74-75, 97, 101, 110-114, 117, 177, 200, 204, 228, 239, 263, 289
- probably ... 235, 239, 250, 257
- problem ... 191, 200, 205, 210, 218-220, 224, 226-229, 234, 239-241, 243, 248, 251-252, 255-256, 263
- problems inherent ... 252
- procedure call ... 49, 77
- process ... 22-37, 39, 41-42, 44-56, 58-64, 66-79, 82-84, 86-87, 89, 92-95, 97-101, 103, 105, 107, 110-115, 117, 120-121, 123-126, 130-132, 135-142, 144, 146-149, 153-154, 162-163, 166, 168-169, 176-177, 180-181, 185-187, 189-191, 194-195, 200, 202, 204, 207, 209, 215-231, 233, 236, 238-241, 244-245, 248-249, 253-256, 277-280, 282-283, 291-294
- process alarm clock ... 70-71, 226
- process controlling ... 22-23, 25-30, 47, 56, 60, 63, 70-71, 78-79, 84, 131, 135-136, 140-141, 146, 148, 153-154, 163, 219-220, 222, 224-225, 229-231, 248, 253, 255, 278, 283, 292-293
- process creation ... 26, 28, 47, 87, 93, 95, 98-99, 216-217
- Process Environment ... 227
- process group ... 22-30, 41, 44-48, 50-51, 56, 58, 62-63, 74-76, 78-79, 92, 98-99, 107, 112, 135-136, 140-141, 148-149, 153-154, 166, 176, 181, 189-190, 204, 209, 215-216, 218-222, 228-230, 253, 255, 278, 280, 283, 291-292
- process group equal ... 27, 44-45, 56, 63, 74-75, 79, 112, 154, 255, 278, 292
- process group ID ... 23-25, 27-30, 41, 44-47, 50-51, 56, 63, 74-76, 78-79, 92, 98-99, 107, 112, 153-154, 204, 209, 218, 221, 228-229, 255, 278, 280, 292
- process group leader ... 23, 25-29, 44, 56, 58, 78-79, 136, 219-220, 255, 278
- process groups ... 22-30, 41, 44-48, 50-51, 56, 58, 62-63, 74-76, 78-79, 92, 98-99, 107, 112, 135-136, 140-141, 148-149, 153-154, 166, 176, 181, 189-190, 204, 209, 215-216, 218-222, 228-230, 253, 255, 278, 280, 283, 291-292
- process ID ... 23-30, 36, 41-42, 44-48, 50-51, 54-56, 62-63, 73-79, 82, 92, 98-99, 107, 111-112, 114-115, 130-131, 153-154, 194, 204, 209, 215, 217-219, 221, 225, 227-229, 239, 255, 278-280, 292-293
- process identification ... 73, 227

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- process image ... 33, 35, 49-52, 217
- process image file ... 49-52, 217
- process lifetime ... 23, 26-28, 86, 233, 241
- process opens ... 26, 31, 34, 42, 50, 55, 89, 92-95, 100, 121, 123, 126, 135-136, 146, 215, 217, 222, 231, 239, 248
- process primitives ... 47, 64, 216
- process prior ... 53, 72, 204, 216-217, 222
- process reading ... 25, 30-31, 36, 123-124, 126, 136-137, 141-142, 236, 239, 245, 248-249, 253, 283
- process termination ... 52-56, 59, 195, 217-219, 221, 223, 291-293
- Process Times ... 82, 231
- process using locking ... 131, 248-249
- processes ... 22-37, 39, 41-42, 44-56, 58-64, 66-79, 82-84, 86-87, 89, 92-95, 97-101, 103, 105, 107, 110-115, 117, 120-121, 123-126, 130-132, 135-142, 144, 146-149, 153-154, 162-163, 166, 168-169, 176-177, 180-181, 185-187, 189-191, 194-195, 200, 202, 204, 207, 209, 215-231, 233, 236, 238-241, 244-245, 248-249, 253-256, 277-280, 282-283, 291-294
- processing ... 22-37, 39, 41-42, 44-56, 58-64, 66-79, 82-84, 86-87, 89, 92-95, 97-101, 103, 105, 107, 110-115, 117, 120-121, 123-126, 130-132, 135-142, 144, 146-149, 153-154, 162-163, 166, 168-169, 176-177, 180-181, 185-187, 189-191, 194-195, 200, 202, 204, 207, 209, 215-231, 233, 236, 238-241, 244-245, 248-249, 253-256, 277-280, 282-283, 291-294
- processor scheduling delays ... 70
- processors ... 70, 260
- PROC_MAX ... 215
- program ... 17-19, 21, 33-34, 36, 38-40, 47, 49, 52, 56, 58, 62, 107-108, 135, 139-141, 155-156, 158, 160-163, 165, 176-177, 181, 183-184, 188-191, 196, 198-200, 202, 206, 208, 215, 217-219, 223, 228, 233, 236, 238-240, 252, 256-257, 259-262, 265, 267-268, 271, 274, 276, 278
- program execution ... 107, 177, 184, 240
- program field ... 165, 233, 239
- programmatic semaphore ... 224
- programmer ... 176, 179, 181-182, 188, 216, 235
- programming ... 17-19, 21, 33-34, 36, 38-40, 47, 49, 52, 56, 58, 62, 107-108, 135, 139-141, 155-156, 158, 160-163, 165, 176-177, 181, 183-184, 188-191, 196, 198-200, 202, 206, 208, 215, 217-219, 223, 228, 233, 236, 238-240, 252, 256-257, 259-262, 265, 267-268, 271, 274, 276, 278
- programming errors ... 33, 135, 208, 223, 236, 265
- prompt ... 38
- proposal ... 185, 194, 198, 206-207, 210-211, 218, 225, 250-251, 257, 259
- Proposals ... 185, 251
- proposed changes ... 187, 209
- protection information ... 169
- provide ... 3-5, 17, 20-22, 25, 27, 30-32, 42, 45, 54, 79, 86-87, 116, 120, 129, 135, 137, 140, 145, 153-154, 163, 169, 175-176, 187, 189-190, 193-194, 196-198, 200-202, 204, 209, 212-213, 216-219, 221-223, 226, 228-229, 233-236, 241-243, 246, 249-250, 254, 258-261, 263-265, 267, 287
- PS1 ... 38, 277
- PS2 ... 38, 277
- ptrace ... 208
- purpose ... 3, 24, 139, 182-183, 194, 197, 206-207, 211, 215, 223, 236, 244, 259, 261
- putenv ... 231
- pwd.h ... 37-38, 167, 173, 209, 289
- pw_dir ... 167
- pw_gid ... 167
- pw_name ... 167
- pw_shell ... 167
- pw_uid ... 167
- qsort ... 155, 192
- quantity ... 232
- queue ... 41, 137, 139, 142-143, 147, 213, 221, 253
- queue_selector ... 151-152
- QUIT character ... 147-148, 254
- race conditions ... 248
- radix ... 38
- raise ... 194, 215
- rand ... 155
- range ... 21-22, 33, 80, 83, 113, 121, 157, 184-185, 196, 211, 214-215, 224, 243, 281-282
- Rationale ... 3, 181, 183-185, 188-192, 194-196, 206-207, 241
- raw mode ... 251
- read ... 22, 24-25, 30-31, 33, 35-36, 43, 59, 62, 89, 91-93, 95, 107-108, 119-120, 122-124, 126-128, 130, 133-

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 134, 136-143, 146-149, 151, 161, 166, 168, 172-173, 208, 212-213, 233, 236, 239, 241-249, 251, 253-254, 263, 265-267, 282-283, 287
- read by group ... 141, 172, 244, 287
- read by others ... 24, 141, 172, 236, 248, 287
- read by owner ... 172, 287
- read operations ... 124, 136, 139, 239, 243, 248-249, 265
- read request ... 30, 33, 122-123, 130, 137-138, 147, 245, 265
- read-only file system ... 28, 31, 94, 97, 99-101, 103-104, 110, 112-113, 115
- readdir ... 87-89, 190, 234-235
- reading-only ... 93
- real group ID ... 25, 28, 50, 73-75, 110, 227-228
- real time ... 5, 70, 83, 178, 221, 226, 230
- Real Time Extensions ... 5, 178, 221, 226, 230
- real user ... 25, 28, 30, 50, 62-63, 73-75, 110, 225, 227-228, 239
- real user ID ... 25, 28, 30, 50, 62-63, 73-75, 110, 225, 227-228, 239
- real-time ... 226
- receipt ... 54, 124, 126, 138-139, 163, 217, 283, 293
- received character ... 137-141, 143, 147-148, 213
- received signal ... 52, 61-64, 130, 253
- receiver ... 46, 63, 144, 146, 213, 217, 246, 253
- receives ... 46, 52, 60-64, 130, 137-141, 143, 146-149, 151, 182, 213, 217, 246, 253, 279
- receiving process ... 46, 60-64, 130, 137, 253, 279
- reception ... 146, 151
- recommendations ... 19, 180, 187, 244
- recommended ... 19, 38, 77, 177, 181, 207, 223, 261, 264, 267, 281
- reference ... 3, 22, 24, 26, 43, 48, 52, 55-56, 58, 64-65, 67-76, 78-79, 83-84, 88, 90-91, 95-97, 99-100, 102-103, 105, 108-109, 111-113, 115, 120-122, 124, 126, 128, 133-134, 149-150, 153, 155, 160-162, 167-168, 170, 173, 183, 185, 189, 191-193, 195, 202, 208, 211, 223, 231, 237, 244, 247, 256-257, 261, 268, 274, 289, 294
- referenced documents ... 191
- region ... 130-132, 247-249
- REGTYPE ... 286
- regular file ... 24, 28, 35, 49, 51, 93, 106-107, 113, 122-125, 129, 172, 204, 228, 236, 238, 240, 248, 263-265, 267, 286, 288
- related standards ... 5, 18, 175-176, 180, 182, 185-186, 188-189, 194-195, 197, 202, 206, 215, 226, 233, 235, 268
- relative offset in bytes ... 130
- release ... 78, 80, 190, 198, 220, 225, 233, 237, 242
- reliable ... 34, 189, 194, 221, 238-239, 261
- reliable queueing ... 221
- relinquish ... 136
- remove ... 22, 88, 100-103, 108, 121, 130-131, 155, 206, 209, 214-215, 237-238, 267, 280
- Remove Directory Entries ... 100, 238
- rename ... 62, 97, 102-104, 155, 190, 193, 236-238, 274
- rename file ... 103-104, 238
- renaming directories ... 238
- renaming dot ... 238
- repeated ... 166, 168, 253
- replaces ... 4, 6, 39, 49, 69, 131, 204, 217, 237, 248, 252, 263, 285, 291
- reposition ... 133, 250
- request ... 30, 33, 35, 52, 57, 66, 70-72, 96-97, 104, 110, 122-123, 125, 127, 130-132, 137-139, 147, 159, 221, 223, 227, 244-246, 248, 265, 281-282
- requested access ... 30, 110, 130-131, 281
- requested time ... 72, 227, 245
- require ... 4, 8, 19-21, 25, 27, 33, 35, 48, 52, 58, 69, 78, 81, 83-85, 92, 96-97, 103-104, 108, 144, 156-157, 176-177, 185-187, 191-195, 197-198, 200, 206-209, 211-213, 216-222, 224-225, 230, 232, 234, 236-240, 242, 244, 246, 248-249, 252, 255-256, 261-262, 264-267, 274-276, 288
- requirement ... 4, 8, 19-21, 25, 27, 33, 35, 48, 52, 58, 69, 78, 81, 83-85, 92, 96-97, 103-104, 108, 144, 156-157, 176-177, 185-187, 191-195, 197-198, 200, 206-209, 211-213, 216-222, 224-225, 230, 232, 234, 236-240, 242, 244, 246, 248-249, 252, 255-256, 261-262, 264-267, 274-276, 288
- Requirements ... 20, 184, 191-192, 196-197
- reserved ... 57-58, 172, 200, 210, 223, 264-265, 286-287, 289
- reset ... 70, 89, 138, 221-222, 259
- resource ... 8, 28, 33, 36, 48, 212, 214, 218,

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 232, 249
- respond ... 239
- response ... 59, 138, 182, 186-187, 209-210, 232
- rest ... 4, 38, 261, 264
- restore signal masks ... 69, 162, 260, 275
- restriction ... 61, 96, 101, 111, 209, 212, 215, 222-223, 229, 232, 235-236, 255, 260
- result ... 36, 49-50, 58, 67-68, 77, 91, 93, 123, 125, 129, 131-134, 168, 185, 187, 198, 204, 227-229, 235, 243, 248-249, 252, 277, 283, 285
- resulting ... 36, 49-50, 58, 67-68, 77, 91, 93, 123, 125, 129, 131-134, 168, 185, 187, 198, 204, 227-229, 235, 243, 248-249, 252, 277, 283, 285
- return types ... 83, 87, 89, 131, 166-167, 191, 207, 209, 230-231, 236, 243, 248
- return value ... 3, 31-32, 48, 51, 54, 56, 63, 65, 67-72, 75-79, 81, 83-86, 89-90, 95-96, 98-99, 101-102, 104, 109-111, 113, 115, 117, 120-123, 126, 129, 131, 133, 150, 152-154, 158, 163, 167-168, 191, 204, 207-208, 227, 230-234, 241, 243, 246, 250, 259, 292-293
- returned argument ... 61, 76, 91, 103, 218, 235-236, 243
- returned value ... 3, 31-32, 48, 51, 54, 56, 63, 65, 67-72, 75-79, 81, 83-86, 89-90, 95-96, 98-99, 101-102, 104, 109-111, 113, 115, 117, 120-123, 126, 129, 131, 133, 150, 152-154, 158, 163, 167-168, 191, 204, 207-208, 227, 230-234, 241, 243, 246, 250, 259, 292-293
- returning zero ... 48, 54, 63, 65, 67-69, 75-76, 79, 89-90, 96, 98-99, 101-102, 104, 109-111, 113, 115, 120, 122-123, 133, 139, 146, 150, 152, 154, 207, 227, 234, 239, 241, 246, 292-293
- reversability ... 261
- rewinddir ... 88-89, 190, 235
- rewinding ... 155, 166, 168
- RFCs ... 185
- right margin ... 6
- right-justified ... 170
- rightmost column ... 212
- risk ... 143, 228
- Ritchie ... 192, 256, 269
- rmdir ... 62, 101-102, 105, 190, 237, 280
- root ... 28, 32, 51, 102, 202, 204-205, 207, 277
- root directory ... 28, 31-32, 51, 102, 202, 204-205, 207, 277
- root file system ... 202, 204
- routine ... 33, 60, 72, 77, 87, 91, 95, 98-99, 161, 168, 203-204, 208, 218, 221, 223, 235, 237, 252, 261, 274
- run time ... 25, 44, 212, 231
- run time match ... 212
- run-time increasable values ... 43, 214
- run-time invariant values ... 41-42, 212, 214
- running process ... 47, 224
- runtime ... 211, 232-233, 240
- runtime facility ... 233, 240
- runtime limits ... 211, 240
- R_OK ... 43, 110
- samefile ... 209
- save ... 45, 162, 194, 217, 259-260, 263
- saved process group ... 28, 50, 74, 255, 278
- saved process group ID ... 28, 50, 74, 255, 278
- saved set-group-ID ... 28
- saved set-user-ID ... 29
- SA_CLDSTOP ... 60, 66, 223, 226
- sa_flags ... 66, 226
- sa_handler ... 66
- sa_mask ... 66-67
- sbrk ... 193
- scanf ... 155
- SCHAR_MAX ... 40
- SCHAR_MIN ... 40
- scheduler ... 70, 177, 225
- scheduling ... 23, 70, 72, 140, 177, 227
- scheduling delays ... 70, 227
- scope ... 3, 17, 20, 177, 183, 189, 194, 197-199, 202, 211, 216, 219, 242, 244, 247, 250, 263
- second ... 17, 23-24, 37, 70, 72, 81-83, 108, 114, 138-139, 157, 202, 220, 226, 230, 234, 249-250, 256-257, 277
- sections ... 3-4, 6, 20-21, 33, 87-88, 135-137, 147, 149, 160, 165, 169, 171, 188-189, 194-195, 204, 208, 211-212, 214, 218-219, 221, 226, 232, 250-252, 258, 264-265, 267, 273, 285, 291
- secure implementation ... 225, 239-240
- secure implementations ... 225, 239-240
- security ... 180, 200, 202, 204, 218, 221, 225, 228-229, 240, 255, 261, 263
- security label ... 225
- security risk ... 228
- seekdir ... 184, 234-235
- seeking ... 36, 122, 124, 133, 161
- seeks ... 36, 122, 124, 133, 161
- SEEK_CUR ... 44, 131, 133

- SEEK_END ... 44, 131, 133, 249
 SEEK_SET ... 44, 131, 133, 212
 select ... 24, 244, 270
 semantic conflicts ... 221
 semantics ... 193, 219-221, 224-226, 241-242, 244
 semaphores ... 224, 247-248
 series ... 169, 177, 179, 285
 session process group leader ... 26, 28-29, 44, 56, 78, 136, 219
 Set Distinguished Process Group ID ... 154, 221, 255
 set file access ... 30, 92, 114, 129, 169, 240
 Set File Access and Modification Times ... 114, 240
 Set File Creation Mask ... 95, 236
 set gid ... 74-75, 172, 287
 set group ID ... 29, 45, 50, 56, 74-76, 78-79, 92, 98-99, 107, 111-112, 153-154, 189, 221, 227-229, 255, 280
 set process group ... 26, 29, 45, 50, 56, 74-75, 78-79, 92, 98-99, 107, 135-136, 153-154, 216, 221-222, 228-229, 255, 280
 set uid ... 74, 172, 228, 263, 287
 set user ... 50, 66, 74, 92, 98-99, 107, 111-112, 189, 227-228, 241, 250
 set-group-ID ... 28
 set-user-ID ... 29
 setbuf ... 155
 setgid ... 23, 28, 62, 74-76, 228, 249, 277
 setgrent ... 166, 274
 sethostid ... 230
 sethostname ... 230
 setjmp ... 155, 162, 194, 227, 244, 260
 setjmp.h ... 162
 setlocale ... 155, 158-159, 211, 257-259
 setpgrp ... 28-29, 62, 64, 78, 136, 153-154, 229
 setpgrp2 ... 28-29, 62, 64, 78, 136, 153-154, 229
 setpwent ... 167-168, 274
 settable parameters ... 142, 254
 setting ... 37, 72, 130, 145-146, 158-159, 222, 249, 252, 258
 setuid ... 23, 28-29, 50, 62, 74, 225, 227-228, 277-278
 Seventh Edition ... 4, 188
 several login names ... 77
 shall ... 4, 6, 19-23, 27, 31-34, 37, 39, 41-45, 47-64, 67-72, 77-80, 82-84, 86-90, 92-94, 96, 98-104, 106-111, 113-115, 117, 119-126, 129-131, 133, 135-143, 146-149, 151-154, 156-158, 162-163, 166-167, 169-173, 176, 196-197, 227-228, 233, 242-246, 248, 263, 277, 279-280, 287-289, 291-293
 shell ... 5, 17, 38, 135, 167, 169, 176-177, 207, 210, 215-216, 219-220, 222-225, 229-230, 253, 255, 277
 short ... 130, 138, 196, 205, 209-210, 283
 short name ... 196
 shortcomings ... 220
 should ... 6, 19-20, 22, 27, 31-33, 45, 50, 56, 77, 101, 107, 138, 149, 163, 166, 172, 177, 187, 191, 195-198, 204, 206-208, 211-219, 222-225, 227-228, 230, 232, 234, 236, 239-240, 243-247, 252-254, 257, 259, 261-263, 266-267, 279, 287-289
 SHRT_MAX ... 40
 SHRT_MIN ... 40
 side effects ... 26, 163
 SIGABRT ... 52, 58, 222, 279
 sigaction ... 50, 55-57, 60, 62, 64-68, 70-72, 78, 95, 124, 126, 133-134, 162-163, 220, 260-261, 275, 292, 294
 sigaddset ... 62, 64-65, 275
 SIGALRM ... 58, 70
 SIGBUS ... 223
 SIGCHLD ... 223, 226
 SIGCLD ... 55, 59-61, 66, 221, 223-224, 226
 SIGCONT ... 56, 59-61, 63, 220, 223-225, 279
 sigdelset ... 62, 64-65, 275
 SIGEMT ... 223
 sigfillset ... 62, 64-65, 275
 SIGFPE ... 58, 61, 223
 sighold ... 275
 SIGHUP ... 44, 56, 58, 141, 219-220, 278
 sigignore ... 275
 SIGILL ... 58, 61, 223
 siginitset ... 62, 64, 225, 275
 SIGINT ... 34, 58, 140, 142, 216, 221
 SIGIOT ... 222-223
 sigismember ... 62, 64-65, 275
 SIGKILL ... 58, 60-61, 163, 220, 222, 224-225, 279
 siglongjmp ... 162, 194, 260, 275
 signal ... 23, 26, 29, 34, 36, 44, 47-48, 50-72, 94, 121-126, 130, 132, 136-138, 140-142, 148-149, 152, 155, 162-163, 183, 189, 191, 193-194, 204, 216-217, 219-227, 229, 235, 244, 249, 253, 255, 260-261, 275, 278-279, 281, 283, 292-294
 signal catching function ... 61, 72, 224, 227, 261
 signal handler ... 34, 66-67, 130, 221,

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 224-225
- signal handling ... 57, 66, 70, 155, 163, 222, 224, 261, 275
- signal interfaces ... 23, 141, 163, 183, 220, 261
- signal mask ... 51, 53, 60, 66-69, 162-163, 220, 223, 226, 260, 275, 292
- signal names ... 57, 136, 183, 220-223, 260
- signal-catching ... 61-62, 66, 69-71, 163, 220-221, 223-224, 227, 276
- signal-catching function ... 61-62, 66, 69-71, 163, 220, 224, 227, 275-276
- signal-catching routine ... 221
- signal.h ... 34, 48, 52-53, 55, 57, 59, 62, 64-70, 72, 94, 96, 122, 132, 136, 149, 162-163, 217, 224, 253, 275, 279, 281, 283, 293-294
- signals ... 23, 26, 29, 34, 36, 44, 47-48, 50-72, 94, 121-126, 130, 132, 136-138, 140-142, 148-149, 152, 155, 162-163, 183, 189, 191, 193-194, 204, 216-217, 219-227, 229, 235, 244, 249, 253, 255, 260-261, 275, 278-279, 281, 283, 292-294
- signals pending ... 48, 51, 60-61, 63-64, 68-69, 163, 216, 220, 223, 226, 279
- sigpending ... 51-52, 62, 65, 68-70, 221, 275
- SIGPIPE ... 36, 58, 126
- sigprocmask ... 51-52, 60, 62, 65-70, 162, 226-227, 260, 275
- SIGQUIT ... 34, 58, 140
- sigrelse ... 275
- sigreturn ... 221
- SIGSEGV ... 58, 61, 223, 279
- sigset ... 275-276
- sigsetjmp ... 162, 194, 260, 275
- sigsetops ... 57, 64, 67-70
- sigset_t ... 57, 64, 66-69, 222
- sigstack ... 221
- SIGSTOP ... 54, 59-61, 163, 223, 279, 293
- sigsuspend ... 60, 62, 65-70, 162, 226-227, 260, 275
- SIGSYS ... 223, 279
- SIGTERM ... 58, 222
- SIGTRAP ... 222-223
- SIGTSTP ... 54, 59-60, 140, 222-223, 279, 283, 293
- SIGTTIN ... 54, 59-60, 136, 222-223, 253, 279, 283
- SIGTTOU ... 54, 59-60, 136, 147-149, 222-223, 253, 279, 283
- SIGUSR1 ... 58, 222-223
- SIGUSR2 ... 58, 222-223
- SIG_BLOCK ... 67
- SIG_DFL ... 50, 57, 60, 66, 163, 221-224, 276
- SIG_HOLD ... 276
- SIG_IGN ... 50, 57, 60-61, 66, 222-224, 276, 279
- SIG_SETMASK ... 68
- SIG_UNBLOCK ... 67
- similar feature ... 240-241
- similar usages ... 159, 196
- simple abnormal termination ... 52, 59
- simple sum ... 288
- sixteen bit problem ... 243
- size ... 34, 87, 91, 106, 126, 130, 133, 142, 144, 146, 169, 195, 198, 210, 215, 217, 225, 230, 234-235, 245, 250-251, 264, 266, 285-289
- size field ... 106, 225, 264, 287-289
- slash ... 24, 26-30, 32, 38, 49, 90, 156, 158, 206-207, 257, 263, 267, 287
- sleep ... 33, 62, 70, 72, 131, 218, 227, 249
- socket ... 241-242, 260
- sophistication ... 232
- sort ... 210, 274
- source code level ... 17-18, 176
- source form ... 232
- source level ... 3, 17-18, 176, 242
- space ... 21-22, 25, 27-28, 35-36, 39, 42, 48, 68-69, 98, 100, 125-126, 156, 198, 210, 213, 235, 238, 244-245, 256, 287
- spawn ... 216
- special ... 18, 22, 24, 27, 29, 32, 35, 38, 46, 58-59, 63, 66, 76, 93, 97, 99, 106-107, 121-122, 135, 138, 140-141, 143, 147-148, 171-172, 184, 189, 201-203, 222, 225, 229, 233, 237, 250, 254, 258-259, 264, 275, 277, 279-280, 283, 286, 288-289
- special characters ... 22, 24, 29, 46, 58-59, 93, 106-107, 135, 138, 140-141, 147-148, 171-172, 225, 254, 280, 283, 288
- special control character functions ... 147-148
- special control characters ... 147-148, 225, 254, 283
- special emphasis ... 18
- special file creation ... 97, 237
- special files ... 22, 24, 27, 29, 35, 93, 97, 99, 107, 121-122, 135, 141, 143, 148, 171, 189, 201-202, 222, 237, 250, 275, 280, 286, 288-289
- special functions ... 18, 22, 27, 35, 122, 140-141, 147-148, 222, 233, 237
- special positioning type ... 250
- specific bit encodings ... 218

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- specific implementation ... 33, 39, 41-43, 59-61, 66, 80, 117, 149, 151, 158-159, 191, 197, 200-202, 204, 209, 212-214, 218, 222, 230, 240-241, 243-244, 247, 250-251, 277, 280, 288, 292-293
- specific interfaces ... 17, 144, 183, 185, 191, 200, 202, 250
- specification ... 3, 29, 44-45, 156, 158, 177-178, 183, 188, 190-191, 196, 198, 208, 215-216, 221, 224, 238, 248, 252, 260, 263, 288-289
- specified file descriptors ... 119
- sprintf ... 155
- SQL ... 177, 179
- srand ... 155
- sscanf ... 155
- Standard ... 3-4, 6, 21, 34, 36, 39-40, 45, 52, 56, 58, 62, 108, 155-156, 158, 160-163, 175-176, 180, 184, 187-194, 196-198, 200, 208-209, 211, 216-217, 219-220, 225-226, 230, 233, 237-239, 241-243, 247-248, 250, 252, 256, 260-261, 263, 267-268, 273-274, 287
- standard allows ... 158, 183, 190, 196, 198, 200, 206, 213, 215, 220, 222, 225, 236-237, 239-242, 249, 252, 257
- standard error ... 32-33, 69, 78, 81, 83-85, 135, 160, 183, 208, 216, 220, 236-237, 239, 241, 244, 246-247, 265, 276, 288
- standard input ... 34, 135, 160, 244
- standard operating system interface ... 3, 17, 25, 175, 181-184, 191-192, 195, 202-203, 205, 230, 252
- standard output ... 135, 160, 288
- standard permits ... 21, 200, 204, 207, 216, 230-231, 236, 240, 243-244, 246-247, 260, 266
- Standard Portable Operating System ... 3, 175
- standard requires ... 19-21, 25, 69, 78, 81, 83-85, 157, 176, 186, 191-193, 197, 206, 208, 211-213, 216-217, 220-222, 224-225, 236-240, 242, 244, 246, 264-267
- Standards Closely Related to the 1003.1 Document ... 176
- standards language ... 3, 17, 21, 34, 36, 39-40, 52, 56, 58, 62, 108, 155-156, 158, 160-163, 176-178, 184, 188-193, 195-196, 211, 219, 226, 234, 237, 239, 254, 256, 261, 268, 274
- START ... 141, 143
- start-up time ... 83
- start/stop input control ... 142-143
- start/stop output control ... 142-143
- stat ... 20-21, 30-32, 42, 52, 62, 95, 99-100, 106, 108-109, 111-112, 173, 201, 233, 239-240, 277, 287-289
- stat structure ... 106, 109, 239, 288
- stat.h ... 106
- state ... 3, 20-21, 28, 30-32, 42, 47, 52, 62, 72, 89, 95, 99-100, 106, 108-109, 111-112, 121, 146, 149; 158, 173, 194, 198, 201, 208, 217, 230, 233, 237, 239-240, 255-256, 259, 261, 268, 273, 277, 287-289
- static data ... 77, 85, 167-168
- status ... 4; 26, 31, 52-55, 92, 106, 108, 124-125, 127-129, 131, 144, 146, 218, 239, 247, 278, 291-293
- status corresponding ... 53-54, 292-293
- status values ... 31, 53-54, 131, 218, 292-293
- stat_loc ... 53-55, 218-219, 291, 293-294
- stderr ... 160, 260
- STDERR_FILENO ... 160
- stdin ... 160, 260
- STDIN_FILENO ... 160
- stdio ... 239, 274
- stdio.h ... 76-77, 84, 160-161
- stdlib.h ... 199-200
- stdout ... 160, 260
- STDOUT_FILENO ... 160
- stime ... 274
- stopped children ... 53-54, 66, 220, 222, 291
- stopped process ... 25, 54, 56, 59-60, 136, 148, 220, 222, 224, 278, 293
- stops ... 25, 53-54, 56, 59-60, 66, 136, 141, 143-144, 146, 148, 220, 222, 224, 278, 291, 293
- store ... 39, 53, 66, 68-69, 80-81, 84, 98, 102, 104, 137, 145-146, 149, 200, 265, 267, 285, 287, 291
- strchr ... 156
- streq ... 156
- strcpy ... 156
- strncpy ... 156
- stream ... 88-90, 148, 151, 160-161, 170, 173, 189, 192-193, 200, 208, 218, 235, 242, 260
- streams ... 88-90, 148, 151, 160-161, 170, 173, 189, 192-193, 200, 208, 218, 235, 242, 260
- strftime ... 156
- Strictly Conforming Application ... 18, 20-21, 41, 43-44, 197-198, 217
- string ... 26, 32, 34-35, 37-39, 41-42, 50, 77, 83-85, 90, 94, 97-98, 100-102, 104,

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 109-110, 112-113, 115, 156, 158-159, 161, 170-171, 230-231, 233, 257-259, 274, 287
- String Handling ... 156, 258
- string terminator ... 230, 233
- strlen ... 156, 234
- strncat ... 156
- strcpy ... 156
- strpbrk ... 156
- strchr ... 156
- strspn ... 156
- strstr ... 156
- strtok ... 156
- struct ... 65, 80, 82, 87-89, 106, 108, 114, 130-131, 145, 149, 166-167, 233-234
- structure ... 17, 20, 22, 28, 31, 43, 57, 66, 80, 82, 87-88, 106, 109, 114, 130, 142, 145, 147, 149, 163, 166-167, 171, 177, 183, 195, 198, 204-205, 210, 216, 222, 225, 230, 233, 238-240, 249, 251, 254, 275, 280-281, 283-284, 286, 288
- structure elements ... 82, 87, 106, 249
- structure members ... 66, 82, 106, 114, 142, 166-167, 230, 240, 280
- st_atime ... 51, 89, 93, 98-99, 106, 108, 119, 123, 240
- st_ctime ... 93-94, 96, 98-99, 101-102, 104, 106, 108, 111, 113-114, 119, 126
- st_dev ... 106
- st_gid ... 106
- st_ino ... 106
- st_mode ... 106-107
- st_mtime ... 93-94, 96, 98-99, 101-102, 104, 106, 108, 119, 126, 240
- st_nlink ... 106, 209
- st_rdev ... 106, 288
- st_size ... 106
- st_uid ... 106
- subject ... 4, 23, 28, 96, 101, 193, 199
- subroutines ... 17
- subscript ... 148
- subscript names ... 148
- subsection ... 3-4, 33, 39, 186, 208, 212
- substantive change ... 210
- success ... 98, 102, 104, 243
- successful call ... 113, 131, 207
- successful completion ... 48, 51, 63, 65, 67-71, 75, 79, 81, 83, 89-90, 93-94, 96, 98-99, 101-102, 104, 109, 111, 113-115, 119-123, 125-126, 131, 133, 150, 152-154, 208
- successful function ... 48, 51, 63, 65, 70-71, 79, 89, 93-94, 96, 98-99, 101-102, 104, 111, 113-115, 119-123, 125-126, 126, 161, 207-208, 246, 248
- successful return ... 48-49, 63, 65, 67-71, 75-76, 79, 81, 83, 89-91, 94, 96, 99, 101, 109, 111, 113, 115, 120-126, 131, 150, 152-154, 161, 207, 246, 248
- sufficient ... 207, 209, 217, 223, 260
- suffix ... 209, 287
- sum ... 33, 82, 217, 288
- summer time ... 156-158, 256-257
- super-user ... 184, 200, 204, 227-228, 236, 238-240, 263
- supplementary group ID ... 24-25, 29, 41, 45, 50, 75-76, 111-112, 204, 228, 240
- supplementary groups ... 24-25, 29, 41, 45, 50, 75-76, 111-112, 189, 204, 228, 240
- support ... 3, 17, 20, 25, 28, 32, 42, 44, 47, 54-56, 59-60, 63, 66, 79, 93, 97, 117, 123, 125, 129, 132, 135-136, 141, 144-146, 149-150, 152-154, 172, 176, 181, 191, 197, 201, 211, 214, 217, 219, 229, 239, 241, 244, 248-249, 251, 256-257, 259-260, 263, 267, 274-275, 288-289, 291-293
- supported languages ... 244, 256-257, 259
- SUSP ... 140-141, 147-148, 283
- suspend ... 25, 52-53, 69-72, 141, 143, 147, 151-152, 227, 230, 249, 255, 292
- suspend process execution ... 25, 52-53, 70-72, 227
- suspended input ... 152
- suspended output ... 141, 143, 151
- suspension ... 70, 72
- suspension time ... 70, 72
- SVID ... 188-189, 209, 233, 235, 237-238, 270, 273-284
- symbolic constant ... 21, 25, 39, 43-45, 57, 77, 84, 86, 110, 116, 133, 135, 183, 195, 215-216, 230, 232, 286
- Symbolic Constants ... 25, 43, 110, 135, 195, 215, 230
- symbolic link ... 205, 262-265, 267
- symbolic name ... 33, 77, 86, 116, 183, 201, 208, 230, 264-265, 283
- SYMTYPE ... 286
- sync ... 274
- synopsis ... 3, 47, 49, 53, 55, 57, 62, 64-65, 67-76, 78-85, 87-88, 90-92, 95-97, 99-100, 102-103, 106, 108, 110-112, 114, 116, 119-122, 124, 127-128, 133, 142, 149, 151, 153-154,

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 158, 160-163, 166-167, 222, 247, 254, 291
- syntax ... 177, 225-226, 237, 247, 257-259
- sys/dir.h ... 233
- sys/stat.h ... 24-25, 31, 52, 63, 93, 95-96, 99-100, 106-109, 111-113, 115, 173, 195, 209, 277, 281, 289
- sys/times.h ... 82
- sys/types.h ... 37, 73-75, 82, 87-88, 92, 95, 97, 99, 106, 108, 111-112, 114, 128, 133, 209
- sys/utsname.h ... 80
- sys/wait.h ... 53, 291
- sysconf ... 44, 85-86, 233, 241, 274, 276
- sysname ... 80
- system ... 3-5, 7, 17, 20, 23, 25, 27-31, 33-38, 42, 47-48, 50, 54-55, 57, 59-61, 63-64, 67-69, 72, 76, 80-83, 85-87, 92, 94, 97-101, 103-106, 110, 112-113, 115, 117, 120, 131-132, 137-138, 143, 156, 158, 165, 169, 172, 175-185, 188-192, 194-198, 200-209, 211, 213, 215-226, 228-233, 236-244, 246-248, 250-252, 256-257, 259-266, 269-271, 273-276, 279, 287-289, 292
- system administration ... 184, 197, 200, 231
- system call ... 27, 82, 183, 194, 203-204, 208, 211, 220, 222, 228-229, 239, 241-242, 247, 252, 279, 287
- system compile time ... 211, 230
- system CPU time ... 82
- system databases ... 165, 257, 261
- system default ... 158, 165, 189, 224, 259
- system documentation ... 3-4, 20, 48, 50, 80, 183, 189
- system identification ... 80, 230
- System III ... 4, 188-190, 203, 219-220, 230, 240-241, 244, 250-251, 269
- System III/V ... 242
- system processes ... 25, 27-31, 33, 47-48, 50, 54-55, 59, 61, 63-64, 82-83, 87, 92, 101, 103, 105, 117, 137, 177, 189, 191, 200, 202, 209, 215, 217, 219, 221-224, 228-229, 231, 233, 240-241, 248, 279, 292
- system programs ... 156, 158, 165, 183, 188, 191, 196, 200, 202, 206, 223, 228, 233, 239-240, 252, 257
- system services ... 17, 47, 97, 201-202, 216
- system start-up time ... 83
- System Time ... 81, 221, 230
- system-imposed ... 33, 35, 48, 51-52, 120, 132, 248
- systems conforming ... 4-5, 20, 156, 169, 172, 185, 192, 215, 221-222, 226, 239, 242-244
- SYS_OPEN ... 215
- S_IRGRP ... 107
- S_IROTH ... 107
- S_IRUSR ... 107
- S_IRWXG ... 107-108
- S_IRWXO ... 107-108
- S_IRWXU ... 107-108
- S_ISBLK ... 107
- S_ISCHR ... 107
- S_ISDIR ... 107
- S_ISFIFO ... 107
- S_ISGID ... 107, 111-112, 238, 240, 281
- S_ISREG ... 107
- S_ISUID ... 62, 107, 111-112, 238, 240, 281
- S_IWGRP ... 107
- S_IWOTH ... 107
- S_IWUSR ... 107
- S_IXGRP ... 107
- S_IXOTH ... 107
- S_IXUSR ... 107
- TABDLY ... 284
- tabs ... 253-254
- tape ... 35, 251, 263, 265-267, 285
- tape mark ... 266
- tapecntl ... 251
- tar ... 185, 190, 262-263, 267, 285
- tar.h ... 286
- tblock ... 285
- tc ... 252
- tcdrain ... 62, 151-152, 284
- tcflow ... 62, 151-152, 284
- tcflush ... 62, 151-152, 284
- tcgetattr ... 46, 62, 149-150, 284
- tcgetpgrp ... 62, 136, 153, 255, 275
- tcgets ... 252
- TCIFLUSH ... 151
- TCIOFF ... 152
- TCIOFLUSH ... 151
- TCION ... 152
- TCOFLUSH ... 151
- TCOOFF ... 151
- TCOON ... 151
- tcsendbreak ... 62, 151-152, 284
- tcsetattr ... 46, 62, 149-150, 284
- tcsetpgrp ... 62, 79, 136, 153-154, 229-230, 255, 275
- tellldir ... 184, 235
- term ... 19, 21-22, 29, 38-39, 138, 182, 184-185, 188, 191-193, 195, 197, 200-204, 206-208, 212, 231, 251, 261, 265, 268, 277
- termcntl ... 251
- terminal ... 22-23, 25-26, 29-31, 38, 41-42,

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 46, 51, 56, 58-59, 63, 77-79, 84-85,
122, 135-144, 146-150, 152-154,
177, 200-201, 208, 213, 222, 225,
229, 231, 248, 250-255, 265, 275-
276, 278, 283-284
- terminal device ... 29, 85, 135, 137, 140-141,
143-144, 213, 231, 248, 251,
253-254
- terminal device file ... 85, 135, 137, 140-141,
248, 253-254
- terminal device name ... 85, 231
- terminal driver ... 25, 225, 253
- terminal file ... 31, 85, 122, 135-137, 140-
141, 143, 148, 150, 152-154, 201,
222, 231, 248, 253-254
- Terminal I/O ... 276, 283
- Terminal Identification ... 84, 231
- terminal input ... 23, 41-42, 135-138, 141-
143, 147, 152, 213, 253, 284
- terminal instead of terminal device ... 231
- terminal interface ... 23, 25, 29, 135-136,
141, 144, 149, 208, 250-252, 255,
275
- terminal parameters ... 136, 149
- Terminate Process ... 55, 219, 221
- terminated child ... 53-54, 59, 82, 278,
292-293
- terminated children ... 54-55, 82, 219-220,
223-224, 292
- terminating process ... 50, 52-55, 59-60, 69,
71, 82, 136, 219-221, 224, 230, 255,
278, 292-293
- termination ... 52-56, 58-59, 195, 217-219,
221, 223, 291-293
- termination consequences ... 56, 219
- terminology ... 17, 19, 196
- termio ... 135, 142, 145, 149, 190, 251, 254,
276, 283-284
- termios ... 135, 142, 145, 149, 190, 251, 254,
276, 283-284
- termios information ... 149, 251
- termios structure ... 142, 145, 149, 251, 254,
283-284
- termios.h ... 46, 142, 149-151, 153
- termios_p ... 145, 149
- test ... 4-5, 43, 65, 106-107, 110, 141, 178,
198, 214-215, 232, 239, 249
- testability ... 212, 215
- testing ... 4-5, 43, 65, 106-107, 110, 141,
178, 198, 214-215, 232, 239, 249
- txt ... 6, 144, 156, 177, 192, 194-195, 238,
260, 276, 280
- Text vs. binary file modes ... 260
- TGEXEC ... 287
- TGREAD ... 287
- TGWRITE ... 287
- this standard ... 3-8, 17-22, 29, 31-33, 37-40,
43, 45, 48, 57, 59-61, 64-65, 69, 78,
81, 83-85, 106, 108, 158, 160, 163,
169, 175-179, 181-186, 188-193,
195-198, 200, 202-204, 207-213,
215-217, 219-223, 225-226, 228-
230, 232-233, 235-245, 247, 249-
250, 252, 254, 256-257, 260-262,
264-267, 273, 275, 285, 288, 291
- TIME ... 138-139, 147-148
- Time of last access ... 106
- Time of last data modification ... 106
- Time of last file status change ... 106
- time remaining ... 28, 60
- time standard ... 7, 31, 45, 81, 157, 176, 186,
202, 211, 221, 231, 239, 241, 248,
252, 256-257
- time zone ... 38, 156-158, 256-257
- time-accounting information ... 82
- time-related fields ... 31, 108-109
- timeout facility ... 249
- timer ... 5, 7-8, 24-25, 28, 31, 37-38, 42, 44-
45, 47-48, 51-52, 55, 59-60, 62,
70-72, 81-83, 106, 108-109, 111,
114-115, 119, 138-139, 147-148,
156-158, 170-171, 176-178, 186,
193, 202, 207-209, 211-212, 215-
216, 221, 226-227, 230-231, 238-
241, 245, 249, 252, 256-257, 261,
277, 280, 288, 294
- timer operations ... 70, 226
- timer value ... 24, 31, 44, 72, 81-83, 114-
115, 119, 147-148, 156-158, 193,
212, 227, 230-231, 245, 288
- times ... 5, 7-8, 24-25, 28, 31, 37-38, 42, 44-
45, 47-48, 51-52, 55, 59-60, 62,
70-72, 81-83, 106, 108-109, 111,
114-115, 119, 138-139, 147-148,
156-158, 170-171, 176-178, 186,
193, 202, 207-209, 211-212, 215-
216, 221, 226-227, 230-231, 238-
241, 245, 249, 252, 256-257, 261,
277, 280, 288, 294
- times.h ... 82, 233
- time_t ... 37, 81, 106, 114, 209, 230, 277,
280
- timing windows ... 216
- title ... 261
- tloc ... 81
- TMAGIC ... 286, 289
- TMAGLEN ... 286
- tmpfile ... 155
- tmpnam ... 155
- tms_cstime ... 47, 51, 82

- tms_cutime ... 47, 51, 82
- tms_stime ... 47, 51, 82
- tms_utime ... 47, 51, 82
- toascii ... 193
- TOEXEC ... 287
- Token Bus ... 178
- Token Ring ... 178
- tolower ... 155
- TOREAD ... 287
- TOSTOP ... 136, 147-148
- total ... 39, 48, 50, 142, 231
- total times ... 231
- toupper ... 155
- TOWRITE ... 287
- trademark ... 3, 5, 181, 185
- traditional function ... 252
- traditional implementations ... 221, 226-227
- trailer ... 171, 266
- trailing null ... 287-288
- translate ... 143, 169, 184, 187, 212, 256
- translation ... 192, 197, 211, 214, 258
- translation and execution environments ... 211
- Translation vs. Execution Environment ... 197
- translator ... 211
- transmission ... 140, 146, 151
- transmitting data ... 143, 151
- transportable archive ... 170, 285
- transportable medium ... 263, 285
- Trial Use ... 6, 186-187, 189-190, 194, 197, 209, 211, 215, 220, 233, 237-238, 252, 263, 267
- troff ... 6
- Truncate flag ... 128
- trust ... 180
- TSGID ... 287
- TSUID ... 287
- TSVTX ... 287
- time_t ... 209
- tyname ... 84-85, 192, 231
- TUEXEC ... 287
- TUREAD ... 287
- TUWRITE ... 287
- TVERSION ... 286
- TVERSLEN ... 286
- type ... 22, 24, 26, 31, 37-38, 57, 60, 64, 66, 80, 82-83, 87-89, 92, 106, 114, 129-132, 142, 160-161, 166-167, 170, 172, 185, 191, 193, 195, 199, 202, 207, 209-210, 220, 222, 225-227, 230-231, 233-234, 236, 240-241, 243-244, 248-252, 260, 262-265, 267, 275, 277, 279-280, 283, 285, 288
- type arguments ... 191, 199, 250-252, 260
- type difference ... 233
- typeflag ... 286, 288
- types.h ... 37, 244, 249
- TZ ... 38, 211
- tzname ... 158
- UCHAR_MAX ... 40
- UID ... 200, 263, 287
- UID_MAX ... 25, 30, 41, 75, 86, 113, 281
- uid_t ... 37, 73-75, 106, 112, 166-167, 209-210, 238, 277
- UINT_MAX ... 40
- ulimit ... 274
- ULONG_MAX ... 40
- umask ... 51-52, 62, 93, 95, 98-100, 236, 277
- umount ... 203, 274
- uname ... 62, 80-81, 190, 230, 286-287, 289
- undefined ... 19, 34, 163, 196-197, 208, 231, 264-265, 285
- undefined results ... 285
- undefined term ... 231
- underscores ... 27, 38
- understanding ... 224, 232-233, 265, 267
- unidirectional sockets ... 242
- unique ... 4, 24-25, 33, 38, 47, 148, 202, 204, 220, 222
- unistd.h ... 20-21, 43-45, 86, 111, 116-117, 133-134, 160, 198, 215, 232, 289
- United States ... 158, 256, 268
- units ... 26, 138, 158, 213, 256, 268, 289
- UNIX ... 3-4, 181-185, 188-189, 200-204, 206, 256
- unlike ... 220, 240-241, 252
- unlink ... 46, 62, 97, 100-101, 103, 105, 108, 236, 238, 280
- Unlock ... 127
- unmasked signals ... 223
- unpredictable behavior ... 157
- unrecoverable error ... 244, 247
- unsigned argument ... 236, 249
- unsigned char ... 142
- unsigned long ... 142, 283
- unsigned offsets ... 249-250
- unsigned short ... 209, 283
- unslept amount ... 72
- unspecified ... 19, 61, 87, 196-197, 219, 234, 251
- unstandardized ... 155, 256
- upper ... 38, 156, 193, 205, 214
- upper case ... 205
- usage ... 21, 25, 155, 159, 182, 195-196, 202, 204, 218, 248, 256, 259-260
- useful applications ... 211, 215, 232, 244, 248
- user ... 4, 17, 23, 25, 28-31, 37-38, 41-42,

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

- 44-46, 48, 50, 62-63, 66, 73-77, 82, 92, 98-99, 106-107, 110-115, 135, 138-139, 165, 167-168, 170, 176, 183, 189-190, 197-198, 200, 204-205, 211, 215, 218-219, 221-222, 225, 227-229, 239-241, 249-250, 252, 255, 261-264, 267, 274, 277-279, 281, 287-289
- User CPU time ... 82
- User CPU time of descendants ... 82
- user ID ... 23, 25, 28-30, 37, 41-42, 44-45, 50, 62-63, 73-75, 77, 92, 98-99, 106-107, 110-115, 165, 167, 189, 215, 218, 225, 227-228, 239-240, 255, 278-279, 288-289
- User ID number ... 167
- User Identification ... 73, 190, 227
- User Name ... 76, 229
- user processes ... 23, 25, 28-30, 42, 48, 50, 62-63, 74-75, 77, 92, 98-99, 107, 111-112, 114-115, 135, 200, 215, 218-219, 225, 228, 239, 241, 249, 255, 278-279
- user security checks ... 255
- user utility ... 264, 267, 287, 289
- ushort ... 209
- USHRT_MAX ... 40
- USTAR ... 267
- ustat ... 62, 238, 274
- utilities ... 5, 135, 155, 169, 176-177, 263, 265-266, 287
- utility ... 5, 169, 173, 176, 191, 193, 207, 214, 263-267, 285, 287-289
- utimbuf ... 114, 281
- utimbuf structure ... 114, 281
- utime ... 46, 62, 108, 114, 240, 281
- utime.h ... 114, 281
- utsname.h ... 80
- valid file descriptor ... 85, 109, 121-122, 124, 126, 132, 134, 150, 152-154, 239, 243
- valid input characters ... 143
- value ... 3, 19-21, 23-25, 28, 30-34, 39, 41-48, 50-51, 53-57, 60, 63-72, 75-86, 89-90, 92-93, 95-96, 98-99, 101-102, 104, 106-107, 109-117, 119-129, 131, 133-134, 138-139, 141, 143-148, 150-154, 156-163, 167-168, 170, 172, 191, 193, 204, 207-209, 211-215, 217-220, 222, 224, 227-234, 237, 240-241, 243, 245-246, 250-251, 254, 258-259, 263-264, 267, 276, 281-282, 284, 286-289, 292-294
- variable ... 22, 32, 37-39, 42, 49-50, 83, 85-86, 91, 116-117, 156, 158-159, 177, 184, 191, 193, 195, 199-200, 207, 210-211, 214, 216, 225-226, 231-233, 240-241, 247, 251, 256-258, 276-277
- variable argument syntax ... 226
- variable errno ... 32, 91, 193, 207
- variable number ... 22, 32, 191, 195, 225, 247
- variable parameter lists ... 225-226
- variant ... 189, 191-192, 200, 205, 252, 292
- VDM ... 179
- vector ... 166
- VEOF ... 148
- VEOL ... 148
- VERASE ... 148
- verb ... 242
- verification suites ... 198, 215
- Verification Testing ... 4-5, 178, 215
- version ... 4, 21, 45, 80, 155-156, 176, 184, 188-190, 200-201, 207, 218-220, 225-226, 228, 230, 232, 234, 240, 242, 244, 246, 250-251, 260, 263-264, 267, 269, 286-289
- vhangup ... 222
- view ... 186, 216
- VINTR ... 148
- VKILL ... 148
- VMIN ... 148
- void ... 55, 57, 66, 88, 162-163, 166-167, 218, 225, 234
- VQUIT ... 148
- VSUSP ... 148
- VTDLY ... 284
- VTIME ... 148
- wait ... 28, 33, 48, 52-56, 62, 69, 71, 82-83, 93, 127, 130, 135, 139-140, 146, 151, 208, 216, 218-219, 221, 223-224, 226, 231, 253, 265, 278, 291-294
- Wait for Process Termination ... 53, 218, 221, 291
- wait.h ... 53, 275
- wait2 ... 28, 33, 48, 52-56, 62, 69, 71, 82-83, 93, 127, 130, 135, 139-140, 146, 151, 208, 216, 218-219, 221, 223-224, 226, 231, 253, 265, 275, 278, 291-294
- waiting ... 28, 33, 48, 52-56, 62, 69, 71, 82-83, 93, 127, 130, 135, 139-140, 146, 151, 208, 216, 218-219, 221, 223-224, 226, 231, 253, 265, 278, 291-294
- waitpid ... 218-219, 291-294
- WERASE ... 253-254

- west ... 156-157, 256
- Wide area Net ... 179
- will ... 4, 6-8, 17, 21, 44-45, 87, 131, 136, 140, 146, 155, 158-159, 163, 173, 176, 184-185, 189, 194, 196-200, 202, 208, 210, 212-213, 218, 220-224, 226, 230-233, 239-240, 243-246, 248-249, 256-259, 262, 265-267, 278, 282, 285, 291-292
- window ... 177, 179, 216, 227
- WNOHANG ... 53-54, 291-293
- word ... 19, 185, 191, 195-197, 206-207, 214-215, 217, 228, 242, 253-254, 258
- WORD_BIT ... 211
- work ... 3-5, 7-8, 23, 30, 32, 37-38, 44, 51, 90-91, 102, 135, 165, 169, 176-179, 181-186, 188-192, 194, 196, 198, 201, 206-207, 209-211, 215, 217, 221, 223, 226, 230, 232-235, 237, 239, 242-244, 246, 248, 250-252, 257, 260-263, 265
- working directory ... 23, 30, 32, 37-38, 44, 51, 90-91, 102, 165, 201, 207, 210, 217, 235
- Working Directory Pathname ... 91, 235
- working documents ... 177-178, 185-186, 188, 190, 261-262
- Working Group ... 4, 7-8, 135, 169, 176, 178, 181-186, 188, 190-192, 194, 196, 206-207, 209-211, 215, 221, 223, 226, 230, 232-235, 237, 239, 242-244, 246, 250-252, 260-263
- would ... 33-34, 48, 53, 72, 89, 94, 96-98, 100, 104, 120-121, 124-126, 130-132, 134, 138-139, 169, 175, 186, 191, 194-195, 197, 200-207, 209, 215-216, 219-221, 224-226, 229-230, 232-236, 241, 243-244, 246-251, 254, 260, 264-266, 276, 282-283, 287, 292
- write ... 22, 24-25, 30-31, 33-36, 43, 58-59, 62, 92-98, 101-104, 107-110, 114-115, 119-120, 123-128, 130-131, 134, 136, 140, 146, 148, 155, 161, 172, 193, 207-208, 211-213, 217-219, 226, 228, 236, 238, 241-251, 253-254, 260, 263, 265-267, 280-283, 285, 287
- write by group ... 136, 148, 172, 218, 244, 285, 287
- write by others ... 125, 172, 236, 248, 287
- write by owner ... 172, 218, 287
- Write requests ... 125
- Writing Characters and Output Processing ... 140, 254
- WUNTRACED ... 53-54, 218, 291, 293
- W_OK ... 43, 110
- X.212 ... 178
- X.25 ... 179
- X.400 ... 179
- X/OPEN ... 7, 180, 186-188, 232, 252, 268
- X3.159-198x ... 21, 34, 36, 39-40, 52, 56, 58, 62, 108, 155-156, 158, 160-163, 176, 184, 188, 190-191, 196, 239, 256, 261, 268, 274
- X3H3.6 ... 179
- X3J11 ... 176, 184, 188, 190-194, 196, 199, 210-211, 214, 219, 244, 257, 259-260
- X3J11 Rationale ... 188, 190, 192, 194, 196
- XCASE ... 284
- X_OK ... 43, 110, 239
- yardstick ... 4
- zero ... 24-25, 27, 30, 32, 45, 47-48, 53-54, 56-58, 62-63, 65, 67-71, 75-77, 79, 84-85, 89-91, 93, 96, 98-102, 104, 107, 109-111, 113, 115, 120, 122-123, 129, 131, 133, 136, 139-140, 142, 146, 150-152, 154, 162, 170-171, 207, 211, 227, 229, 231, 234, 239, 241, 246, 257, 275, 285, 287-289, 292-293
- zero filled ... 285
- zero-valued bits ... 151, 284
- _asm_builtin_atoi ... 200
- _exit ... 44-45, 52-53, 55-56, 58-59, 61, 78, 193, 218-219, 278, 292, 294
- _longjmp ... 260
- _PC_DIR_DOTS ... 116
- _PC_FCHR_MAX ... 116
- _PC_GROUP_PARENT ... 116
- _PC_LINK_DIR ... 116
- _PC_LINK_MAX ... 116
- _PC_MAX_CANON ... 116
- _PC_NAME_MAX ... 116
- _PC_PIPE_BUF ... 116
- _POSIX_CHOWN_RESTRICTED ... 112, 116
- _POSIX_CHOWN_SUP_GRP ... 112, 116
- _POSIX_DIR_DOTS ... 88, 98, 102-103, 116
- _POSIX_EXIT_SIGHUP ... 56, 86
- _POSIX_GROUP_PARENT ... 92, 98-99, 116
- _POSIX_JOB_CONTROL ... 25, 86, 135
- _POSIX_KILL_SAVED ... 62, 86, 279
- _POSIX_LINK_DIR ... 96, 100, 116
- _POSIX_NO_TRUNC ... 32, 35, 51, 90, 94, 97-98, 100-102, 104, 109-110,

UNAPPROVED DRAFT. All Rights Reserved by IEEE.

Do not specify or claim conformance to this document.

Acknowledgements

We wish to thank the following organizations for donating significant computer, printing, and editing resources to the production of this standard: /usr/group, Amdahl Corporation, Digital Equipment Corporation, and UniSoft Corporation.

Also we wish to thank the organizations employing the members of the Working Group and the Balloting Group for both covering the expenses related to attending and participating in meetings, and donating the time required both in and out of meetings for this effort.

Editor's Note: This list will be included in the final printed standard.

Company...

Company...

Company...

Company...

