The POSIX.1 Standard

A Programmer's Guide

Fred Zlotnick

The POSIX.1 Standard

A Programmer's Guide

Fred Zlotnick

Mindcraft, Inc.

Palo Alto, California

The Benjamin/Cummings Publishing Company, Inc.

Redwood City, California • Menlo Park, California • Reading, Massachusetts
New York • Don Mills, Ontario • Wokingham, U.K. • Amsterdam
Bonn • Sydney • Singapore • Tokyo • Madrid • San Juan

Sponsoring Editor: John Thompson

Production Supervisor: Laura Kenney

Freelance Project Management: Gary Palmatier

Copy Editor: Anna Huff

Composition and Illustration: Ideas to Images

Cover illustration: *Rainbow Kite* © 1991 Mr. Screens

Library of Congress Cataloging-in-Publication Data

Zlotnick, Fred.
The POSIX.1 standard : a programmer's guide / Fred Zlotnick.
p. cm.
Includes index.
ISBN 0-8053-9605-5

1. UNIX (Computer file) 2. Operating systems (Computers)—Standards—United States. I. Title. II. Title: POSIX dot one standard. III. Title: POSIX point one standard.
QA76.76.063Z57 1991
005.4'3—dc20 91-9770
CIP

2 3 4 5 6 7 8 9 10 -DO- 95 94 93 92 91

The Benjamin/Cummings Publishing Company, Inc.
390 Bridge Parkway
Redwood City, Califomia 94065

Preface

About POSIX

The UNIX® system was 10 years ahead of its time, but it is now more than 20 years old. This is not to say that UNIX is now out of date. On the contrary, it has become the dominant operating system for much of the computing industry. It is a de facto standard.

The fact that UNIX systems run on so many different types of hardware has made it possible for application programmers to write portable programs that are not limited to a single manufacturer's platform. However, the widespread acceptance of UNIX has been accompanied by a number of problems, chief among them a proliferation of different, incompatible versions. In fact, the UNIX system does not represent one standard but several, all mutually incompatible. The POSIX standardization effort began as an attempt to deal with this problem by establishing an official standard that, as far as possible, is compatible—*from the program's point of view*—with the core of most historical UNIX implementations. The purpose is to make portable programs possible.

This book deals with the POSIX 1003.1-1990 standard. That standard—commonly referred to as POSIX.1 (pronounced "pah-zix dot 1") or more informally as "dot 1"—describes an operating system interface for C language programs. It is based on a number of documents; the *1984 /usr/group Standard* is its most direct ancestor. Most of the interface descriptions in POSIX can, with slight variations, be found in one of two documents that describe implementations of UNIX systems: the *System V Interface Description (SVID)* published by AT&T and the *4.3 Berkeley Software Distribution (4.3BSD) Manuals.* In general, the philosophy of the 1003.1 committee was to adhere to existing UNIX system interfaces unless there was a good reason to do otherwise.

The official name of the POSIX.1 standard is ISO/IEC IS 9945-1:1990. The International Standards Organization (ISO) and International Electrotechnical Commission (IEC) jointly oversee international computer standards. In the United States, the American National Standards Institute (ANSI) and the Institute of Electrical and Electronics Engineers (IEEE) are

responsible for computer standards, and ANSI/IEEE Std. 1003.1-1990 is another official name for the POSIX.1 standard. We simply use the informal name POSIX.1 or (even more simply) just POSIX when no confusion can arise.

The "-1" in 9945-1:1990 and the ".1" in POSIX.1 both refer to the fact that other operating system interface standards are being developed to cover areas not in the purview of the current standard. These standards are under development by IEEE committees with numbers like 1003.2 (developing POSIX.2). The name *POSIX* is almost an acronym for Portable Operating System Interface. The "IX" suffix is traditional. The name *POSIX* was suggested by Richard Stallman.

All major vendors of the UNIX system either are delivering POSIX.1 conforming systems now or are committed to doing so. Such systems include or will include OSF/1, System V, and BSD. Vendors of some other major operating systems that are not based on the UNIX system, including VMS and OS/2, have also committed to POSIX.1 conformance. Thus, the POSIX effort has moved well beyond the world of UNIX systems in which it originated. Each POSIX.1 implementation will also contain vendor-specific extensions to POSIX, either for compatibility with historical versions or as "value-added" features. If an application programmer is to avoid implementation dependence, he or she must make careful use of the standard's features and avoid nonportable vendor-supplied features.

Goals

The POSIX.1 standard makes it possible to write application programs that are portable across a wide variety of systems and architectures. The goal of the book is to show you how to take advantage of that possibility. Our specific goals are to:

- Explain the syntax and semantics of the 203 C functions and macros supported by the POSIX.1 standard and the data structures that support them.
- Give an understanding of the concepts that you need to use these interfaces and data structures effectively.
- Show how other standards and specifications interact with POSIX.1.
- Show how you can use the POSIX.1 standard to package a portable application, including source code and supporting files.
- Explain the limits of the POSIX.1 standard.

The principal goal of the book is to teach programmers how to write programs that will run under any implementation of POSIX.1—Strictly Conforming POSIX.1 Applications, in the language of the standard. Note that POSIX.1

explicitly addresses application programs only. Interfaces needed for system administration or system programming are not supported.

Audience

This book is written for computer professionals and for students of operating systems.

- Software engineers can only hope to write portable application programs by adhering to the POSIX.1 standard.
- Software engineers who are implementing systems must take POSIX.1 as a partial specification.
- Software engineering managers need to know the possibilities and limits of portable programming using POSIX.1 to understand what degree of portability their programming teams can reasonably be expected to achieve.
- Managers in charge of hardware procurements must understand what vendors do and do not guarantee when they claim to be "POSIX conforming."
- Students should know the features guaranteed by an international operating system interface standard.

Features

This book includes the following features: prototypes for all functions; sample code; appendices for reference; exercises; and a glossary.

Prototypes for All Functions

In the chapters and appendices of this book, C functions are described using the prototype format of the ANSI C standard. Even if you are not familiar with the prototype format, you should be able to understand the code. Here is an example: the C library function *strcpy()* would be described like this in "old style":

```
char *strcpy(s1, s2)
char *s1, *s2;
```

A prototype for this function is written with the argument list, including type information, inside the parentheses:

```
char *strcpy(char *s1, const char *s2);
```

The names of the parameters are optional. Thus, an equivalent way to write this prototype is:

```
char *strcpy(char *, const char *);
```

The reserved word `const` in front of the second argument is an addition to the language by the ANSI C committee. (It is borrowed from the C++ language.) It indicates that the object pointed to by the second argument cannot be modified by a call to the *strcpy()* function. The absence of this qualifier in front of the first argument indicates that the object pointed to by the first argument can be modified by the function.

As a special case, if a function has no arguments, its ANSI C prototype is written with a parameter list consisting of the single reserved word `void`. For example:

```
char *getlogin(void);
```

There is a special notation for prototypes of functions that take a variable number of arguments. The fixed arguments, if any, are declared as in other prototypes. In place of the variable arguments, you put an ellipsis (. . .). For example, a prototype for *fprintf()* is:

```
int fprintf(FILE *stream, char *format, ...);
```

The identifiers `stream` and `format` also can be omitted:

```
int fprintf(FILE *, char *, ...);
```

Every POSIX.1 function is associated with a header. (The terms *header file* and *include file* are obsolete; see Chapter 1, Section 1.2.) When a prototype for a function is presented, the header in which its prototype appears on C standard systems is also given, as well as all other headers that should be included when that function is used.

Sample Code

Most functions that have been introduced by the POSIX.1 committee, or with semantics that have changed significantly from UNIX systems, are used in examples in the text. These examples illustrate how the functions are intended to be used.

Appendices for Reference

The body of this book is written in a "how-to" format. For those who wish to use the book as a reference, appendices are provided.

- Appendix A gives, for each POSIX.1 function that is not imported from the C standard, the function's headers, prototype, return values, possible `errno` values, and a brief description of the function's semantics.

- Appendix B does the same for the POSIX.1 functions that come from the C standard.
- Appendix C describes the 37 portable values of `errno` that are supported by POSIX.1.
- Appendix D lists the headers specified by either POSIX.1 or the C standard and the reserved name-space associated with each header. (See Chapter 1, Section 1.6.) This name-space includes functions, macros, typedefs, structures, and external variables.
- Appendix E lists those POSIX.1 functions that can be safely invoked from a signal-catching function.
- Appendix F gives references to other POSIX standards and proposed standards and to related documents.

Exercises

Exercises are provided at the end of most chapters. Every reader should try some of them to test his or her understanding. They can also be used in a classroom setting.

Glossary

A glossary defining some of the most important terms is included.

About Portability and This Book

Programmers face two different classes of problems when trying to write portable programs. One class deals with nonportabilities intrinsic to the language. For example, it's easy to write C programs that make implicit assumptions about byte order within integers, about the relative sizes of pointers and integers, or about the layout of fields within structures. Any of these assumptions can render the program unportable. One might term these *internal* portability issues. They arise in all programming languages, although C is particularly vulnerable to them. Internal portability has to do with the relationship of the program's own code to hardware; it is under the complete control of the programmer.

Another class of problems has to do with the choice of external interfaces (functions and macros) that a program uses, the semantics that the program assumes for these interfaces, and the types of arguments that the program passes and return values that it expects. These might be termed *external* portability issues. They deal with the code invoked by, but external

to, the program: the libraries and system calls that the program depends on. Problems of external portability arise in many programming languages, but again C is particularly vulnerable, because of its heavy dependence on libraries. External portability is only under the control of the programmer if the external interfaces are standardized. Without standards, external portability is impossible.

This book deals with external portability in the context of POSIX conforming systems. If you've tried to move C programs from one UNIX system to another you may have noticed that, while some programs compile without change and behave identically, others either don't compile or don't run in quite the same way. Given the differences among UNIX systems, this is unavoidable. In principle, if you write a Strictly Conforming POSIX.1 Application program it should compile on all POSIX.1 systems (or on none) and should have, within certain constraints, the same semantics on all of them.

Structure

Chapter 1 presents an overview of POSIX.1 concepts. This is the sort of material that programmers like to skip, to get to the "real code", but in fact it's essential in order to understand the issues that shaped the standard. Chapters 2 through 8 present the POSIX.1 C-language application programming interface (API). The API is the set of programming interfaces specified in the standard. Chapter 9 describes the data interchange formats specified by the standard. Chapter 10 describes the current state of proposed future extensions to POSIX.1. Chapter 11 discusses related standards, draft standards, and specifications, including the work of other POSIX groups and the *X/Open Portability Guide*. Finally, Chapter 12 discusses some general portability considerations for C programs. The six appendices were described above.

Acknowledgments

Only someone who has authored a book can know how much the book depends on the efforts of others. I have been fortunate to have the assistance of able reviewers, editors, and colleagues, without whose help this book could not have been written. Clarke Echols of Hewlett-Packard, Randolph Bentson of Colorado State University, Mark Sobell of Sobell Associates, and Claudia DeBlauw of Mindcraft all reviewed early portions of the manuscript. Robert Bismuth of DEC and Kathy Bohrer of IBM read portions of later drafts of the manuscript and made many valuable suggestions. Jim Isaak of DEC read two complete drafts of the manuscript, corrected many errors, and provided valuable advice and suggestions. I thank all of these people for their efforts. I also have had the good fortune to work with a very

able editorial and production staff at Benjamin/Cummings: Alan Apt, John Thompson, Vivian McDougal, and Laura Kenney guided me through the complex process that ends in a published book. Anna Huff and Gary Palmatier did a thorough, capable job of copy editing and production.

I owe a special debt to my colleagues at Mindcraft, especially Bruce Weiner and Chuck Karish. Bruce introduced me to POSIX and to the standards way of thinking. Chuck read several drafts of this book with great care, pointing out many errors and making numerous worthy suggestions. He also helped me to read and interpret the POSIX.1 standard, with an attention to detail and subtlety that was invaluable. To all of these people I give sincere thanks.

Despite all this help, it is possible (and perhaps inevitable) that errors remain in the text. Of course, the responsibility for such errors is mine alone. Any reader who finds errors, or has suggestions, should notify me either through Benjamin/Cummings or, if desired, by electronic mail over the Internet. I can be reached at the Internet address *fred@mindcraft.com*.

Finally, I thank my wife, Linda Garfield, for giving me encouragement and putting up with the trials that are known only to authors' spouses, and my sons, Ben and Micah, who also encouraged me and (almost always) left me time and space to write when I asked them to, even when they were beset by the urgent needs of adolescence. And thanks, Flash, for curling up at my feet while I wrote.

Brief Contents

Chapter 1: The POSIX Environment 1

Chapter 2: Process and System Attributes 29

Chapter 3: Files and Directories 53

Chapter 4: Input and Output 83

Chapter 5: Signals 105

Chapter 6: Process Creation and Synchronization 135

Chapter 7: Controlling Terminal Devices 157

Chapter 8: ANSI C Standard Functions 177

Chapter 9: Data Interchange Formats 207

Chapter 10: Proposed Revisions to POSIX.1 227

Chapter 11: Related Standards 241

Chapter 12: General C Portability Considerations 263

Appendices:

A: POSIX.1 Functions 277

B: ANSI C Functions in POSIX.1 307

C: Error Numbers 331

D: Headers and Their Contents 337

E: Signal-Safe Reentrant Functions 347

F: Access to Standards 349

References 353

Glossary 355

Index 363

Detailed Contents

Chapter 1: The POSIX Environment 1

1.1 Our Goal 2
- 1.1.1 Implementation Conformance 2
- 1.1.2 Application Conformance 4
- 1.1.3 Our Goal, Restated 5

1.2 The POSIX.1 Environment 6

1.3 Some Differences between UNIX and POSIX Systems .. 8

1.4 Configuration Options 13

1.5 Determining Configuration Values during Execution 16

1.6 Standard Types 20

1.7 Name-Space Pollution 21

1.8 Environment Strings 25

Chapter 2: Process and System Attributes 29

2.1 Determining Current Process Attributes 29
- 2.1.1 BSD Job Control Concepts 30
- 2.1.2 System V Process Groups 31
- 2.1.3 POSIX Process Groups, Sessions, and Controlling Terminals 31

2.2 Process User and Group IDs 35
- 2.2.1 Supplementary Group IDs 38

2.3 Who and Where Am I? 39

2.4 System Databases and Security 42

2.5 Current Working Directory 45
2.6 Environment Strings 46
2.7 Process Times 49
2.8 System Time 49
2.9 System Name 50

Chapter 3: Files and Directories 53

3.1 Pathname Resolution 54
3.2 Determining File Characteristics 55
3.2.1 File Access Permission 58
3.3 File Descriptors and Open File Descriptions 60
3.4 Regular Files 61
3.5 Directories 65
3.6 Pipes 67
3.7 FIFO Special Files 70
3.8 Block and Character Special Files 74
3.9 Controlling File Attributes 75
3.10 Renaming Files 79

Chapter 4: Input and Output 83

4.1 Controlling Open File Descriptions 83
4.2 Controlling File Descriptors 87
4.3 Reading Regular Files 89
4.4 Reading Special Files 91
4.5 Writing Regular Files 92
4.5.1 A Simple Example 94
4.5.2 I/O Synchronization 94
4.6 Writing Special Files 97
4.7 File Locking 98
4.7.1 File Locking and Deadlocks 100

Chapter 5: Signals 105

5.1 Review of Signal Concepts and Implementation 105
5.1.1 The C Keyword `volatile` 109
5.2 The Unreliability of UNIX Signals 110
5.3 Signal Data Structures in POSIX.1 111
5.4 Establishing Signal Actions in POSIX.1 114
5.5 Blocking Signals 116
5.5.1 Actions for Blocked Signals 117
5.6 Special Considerations for Job Control Signals 117
5.7 Sending Signals 119
5.8 Scheduling and Waiting for Signals 121
5.9 Signals and Reentrancy 124
5.10 Signals and Non-Local `Gotos` 126

Chapter 6: Process Creation and Synchronization 135

6.1 Process Creation 135
6.1.1 Handling *fork()* Failure 140
6.2 Program Execution 143
6.3 Synchronizing with Termination of a Child Process 146
6.3.1 Interpreting Child Status 149
6.3.2 Advantages of *waitpid()* over *wait()* 151
6.3.3 Interactions between *wait()* and `SIGCHLD` 152
6.4 Process Termination 152

Chapter 7: Controlling Terminal Devices 157

7.1 Controlling Terminals 157
7.2 Input Processing 158
7.2.1 Special Characters 159
7.2.2 Canonical and Noncanonical Modes 160

7.3 The `termios` Data Structure 162
7.4 Controlling Terminal Attributes 167
7.4.1 Errors in Setting Terminal Attributes 170
7.5 Line Control 172
7.6 Terminal Access and Job Control 173

Chapter 8: ANSI C Standard Functions 177

8.1 Prototypes and Headers 178
8.1.1 Headers in ANSI C and POSIX 180
8.2 Stream I/O 182
8.2.1 File Handles 187
8.2.2 Which Kind of I/O Should You Use? 187
8.3 Internationalization 194
8.3.1 Locale Categories 195
8.3.2 Using Locales 197
8.4 Time Functions 198
8.4.1 Time Zones and Daylight Savings Time 199
8.4.2 More about Time Functions 201

Chapter 9: Data Interchange Formats 207

9.1 Packaging Applications 208
9.1.1 Packaging Source Files 209
9.1.2 Packaging Binary Executable Files 211
9.1.3 Packaging Data Files 212
9.1.4 Pathnames 215
9.2 Extended `tar` Format 215
9.2.1 Restoring Extended `tar` Archives 221
9.3 Extended `cpio` Format 221
9.3.1 Restoring Extended `cpio` Archives 224
9.4 Future Directions 224

Chapter 10: Proposed Revisions to POSIX.1 227

10.1 Proposed New Interfaces 227

10.1.1 Symbolic Links 228
10.1.2 Changing Attributes of Open Files 231
10.1.3 Clarification of *getgroups()* and Supplementary Groups 232
10.1.4 Setting Effective User and Group IDs 233
10.1.5 Manipulating Environment Variables 233
10.1.6 Input and Output 234
10.1.7 Traversing File Trees 235
10.1.8 Message Catalogues and Internationalization 236
10.1.9 New Feature Test Macro 238

10.2 Proposed Language-Independent Interface 238

Chapter 11: Related Standards 241

11.1 The POSIX.1 FIPS 242

11.1.1 Portable Application Programs and the POSIX.1 FIPS 244

11.2 The TCOS Project 244

11.3 Interactions with 1003.2 246

11.3.1 Name-Spaces 247

11.4 Other POSIX Standards 252

11.4.1 The POSIX Guide: 1003.0 254
11.4.2 Shell and Tools: 1003.2 254
11.4.3 Verification of Conformance: 1003.3 255
11.4.4 Real-Time Systems: 1003.4 256
11.4.5 Other POSIX Committees 258
11.4.6 The 1201 Committee 260

11.5 The *X/Open Portability Guide* 261

Chapter 12: General C Portability Considerations 263

12.1 What C Does Not Guarantee 263

12.1.1 Questions of Sign and Sign Extension 263