PREFACE

Subject

In this book, I describe the Linux programming interface—the system calls, library functions, and other low-level interfaces provided by Linux, a free implementation of the UNIX operating system. These interfaces are used, directly or indirectly, by every program that runs on Linux. They allow applications to perform tasks such as file I/O, creating and deleting files and directories, creating new processes, executing programs, setting timers, communicating between processes and threads on the same computer, and communicating between processes residing on different computers connected via a network. This set of low-level interfaces is sometimes also known as the *system programming* interface.

Although I focus on Linux, I give careful attention to standards and portability issues, and clearly distinguish the discussion of Linux-specific details from the discussion of features that are common to most UNIX implementations and standardized by POSIX and the Single UNIX Specification. Thus, this book also provides a comprehensive description of the UNIX/POSIX programming interface and can be used by programmers writing applications targeted at other UNIX systems or intended to be portable across multiple systems.

Intended audience

This book is aimed primarily at the following audience:

- programmers and software designers building applications for Linux, other UNIX systems, or other POSIX-conformant systems;
- programmers porting applications between Linux and other UNIX implementations or between Linux and other operating systems;
- instructors and advanced students teaching or learning Linux or UNIX system programming; and
- system managers and "power users" wishing to gain a greater understanding of the Linux/UNIX programming interface and of how various pieces of system software are implemented.

I assume you have some prior programming experience, but no previous system programming experience is required. I also assume you have a reading knowledge of the C programming language, and know how to use the shell and common Linux or UNIX commands. If you are new to Linux or UNIX, you will find it helpful to read the programmer-oriented review of fundamental concepts of Linux and UNIX systems in Chapter 2.

> The standard tutorial reference for C is [Kernighan & Ritchie, 1988]. [Harbison & Steele, 2002] goes into even more detail on C, and includes coverage of changes introduced with the C99 standard. [van der Linden, 1994] is an alternative look at C that is both highly amusing and instructive. [Peek et al., 2001] provides a good, brief introduction to using a UNIX system.
>
> Throughout this book, indented small-font paragraphs like these are used for asides containing rationale, implementation details, background information, historical notes, and other topics that are ancillary to the main text.

Linux and UNIX

This book could have been purely about standard UNIX (that is, POSIX) system programming because most features found on other UNIX implementations are also present on Linux and vice versa. However, while writing portable applications is a worthy goal, it is also important to describe Linux extensions to the standard UNIX programming interface. One reason for this is the popularity of Linux. Another is that the use of nonstandard extensions is sometimes essential, either for performance reasons or to access functionality that is unavailable in the standard UNIX programming interface. (All UNIX implementations provide nonstandard extensions for these reasons.)

Therefore, while I've designed this book to be useful to programmers working with all UNIX implementations, I also provide full coverage of programming features that are specific to Linux. These features include:

- *epoll*, a mechanism for obtaining notification of file I/O events;
- *inotify*, a mechanism for monitoring changes in files and directories;
- capabilities, a mechanism for granting a process a subset of the powers of the superuser;

- extended attributes;
- i-node flags;
- the *clone()* system call;
- the `/proc` file system; and
- Linux-specific details of the implementation of file I/O, signals, timers, threads, shared libraries, interprocess communication, and sockets.

Usage and organization

You can use this book in at least two ways:

- As a tutorial introduction to the Linux/UNIX programming interface. You can read the book linearly. Later chapters build on material presented in earlier chapters, with forward references minimized as far as possible.
- As a comprehensive reference to the Linux/UNIX programming interface. An extensive index and frequent cross-references allow topics to be read in random order.

I've grouped the chapters of this book into the following parts:

1. *Background and concepts*: history of UNIX, C, and Linux and overview of UNIX standards (Chapter 1); a programmer-oriented introduction to Linux and UNIX concepts (Chapter 2); and fundamental concepts for system programming on Linux and UNIX (Chapter 3).
2. *Fundamental features of the system programming interface*: file I/O (Chapter 4 and Chapter 5); processes (Chapter 6); memory allocation (Chapter 7); users and groups (Chapter 8); process credentials (Chapter 9); time (Chapter 10); system limits and options (Chapter 11); and retrieving system and process information (Chapter 12).
3. *More advanced features of the system programming interface*: file I/O buffering (Chapter 13); file systems (Chapter 14); file attributes (Chapter 15); extended attributes (Chapter 16); access control lists (Chapter 17); directories and links (Chapter 18); monitoring file events (Chapter 19); signals (Chapter 20 to Chapter 22); and timers (Chapter 23).
4. *Processes, programs, and threads*: process creation, process termination, monitoring child processes, and executing programs (Chapter 24 to Chapter 28); and POSIX threads (Chapter 29 to Chapter 33).
5. *Advanced process and program topics*: process groups, sessions, and job control (Chapter 34); process priorities and scheduling (Chapter 35); process resources (Chapter 36); daemons (Chapter 37); writing secure privileged programs (Chapter 38); capabilities (Chapter 39); login accounting (Chapter 40); and shared libraries (Chapter 41 and Chapter 42).
6. *Interprocess communication (IPC)*: IPC overview (Chapter 43); pipes and FIFOs (Chapter 44); System V IPC—message queues, semaphores, and shared memory (Chapter 45 to Chapter 48); memory mappings (Chapter 49); virtual memory operations (Chapter 50); POSIX IPC—message queues, semaphores, and shared memory (Chapter 51 to Chapter 54); and file locking (Chapter 55).

7. *Sockets and network programming*: IPC and network programming with sockets (Chapter 56 to Chapter 61).
8. *Advanced I/O topics*: terminals (Chapter 62); alternative I/O models (Chapter 63); and pseudoterminals (Chapter 64).

Example programs

I illustrate the use of most of the interfaces described in this book with short, complete programs, many of which are designed to allow you to easily experiment from the command line to see just how various system calls and library functions work. Consequently, this book contains a lot of example code—around 15,000 lines of C source code and shell session logs.

Although reading and experimenting with the example programs is a useful starting point, the most effective way to consolidate the concepts discussed in this book is to write code, either modifying the example programs to try out your own ideas or writing new programs.

All of the source code in this book is available for download from the book's web site. The source code distribution also includes many additional programs that don't appear in the book. The purpose and details of these programs are described in comments in the source code. *Makefiles* are provided for building the programs, and an accompanying `README` file gives further details about the programs.

The source code is freely redistributable and modifiable under the terms of the GNU Affero General Public License (Affero GPL) version 3, a copy of which is provided in the source code distribution.

Exercises

Most chapters conclude with a set of exercises, some of which are suggestions for various experiments using the provided example programs. Other exercises are questions relating to concepts discussed in the chapter, and still others are suggestions for programs you might write in order to consolidate your understanding of the material. You'll find solutions to selected exercises in Appendix F.

Standards and portability

Throughout this book, I've taken special care to consider portability issues. You'll find frequent references to relevant standards, especially the combined POSIX.1-2001 and Single UNIX Specification version 3 (SUSv3) standard. You'll also find details about changes in the recent revision of that standard, the combined POSIX.1-2008 and SUSv4 standard. (Because SUSv3 was a much larger revision, and it is the UNIX standard that is in most widespread effect at the time of writing, discussions of standards in the book are generally framed in terms of SUSv3, with notes on the differences in SUSv4. However, you can assume that, except where noted, statements about specifications in SUSv3 also hold true in SUSv4.)

For features that are not standardized, I indicate the range of differences on other UNIX implementations. I also highlight those major features of Linux that are implementation-specific, as well as minor differences between the implementation of system calls and library functions on Linux and other UNIX implementations. Where a feature is not indicated as being Linux-specific, you can normally assume that it is a standard feature that appears on most or all UNIX implementations.

I've tested most of the example programs presented in this book (other than those that exploit features that are noted as being Linux-specific) on some or all of Solaris, FreeBSD, Mac OS X, Tru64 UNIX, and HP-UX. To improve portability to some of these systems, the web site for this book provides alternative versions of certain example programs with extra code that doesn't appear in the book.

Linux kernel and C library versions

The primary focus of this book is on Linux 2.6.*x*, the kernel version in widest use at the time of writing. Details for Linux 2.4 are also covered, and I've indicated where features differ between Linux 2.4 and 2.6. Where new features appear in the Linux 2.6.*x* series, the exact kernel version number of their appearance (e.g., 2.6.34) is noted.

With respect to the C library, the main focus is on the GNU C library (*glibc*) version 2. Where relevant, differences across *glibc* 2.*x* versions are noted.

As this book was heading to press, Linux kernel version 2.6.35 had just been released, and *glibc* version 2.12 had been recently released. This book is current with respect to both of these software versions. Changes that occur in the Linux and *glibc* interfaces after publication of this book will be noted on the book's web site.

Using the programming interface from other languages

Although the example programs are written in C, you can use the interfaces described in this book from other programming languages—for example, compiled languages such as C++, Pascal, Modula, Ada, FORTRAN, D, and scripting languages such as Perl, Python, and Ruby. (Java requires a different approach; see, for example, [Rochkind, 2004].) Different techniques will be required to obtain the necessary constant definitions and function declarations (except in the case of C++), and some extra work may be needed to pass function arguments in the manner required by C linkage conventions. Notwithstanding these differences, the essential concepts are the same, and you'll find the information in this book is applicable even if you are working in another programming language.

About the author

I started using UNIX and C in 1987, when I spent several weeks sitting in front of an HP Bobcat workstation with a copy of the first edition of Marc Rochkind's *Advanced UNIX Programming* and what ultimately became a very dog-eared printed copy of the C shell manual page. My approach then was one that I still try to follow today, and that I recommend to anyone approaching a new software technology: take the time to read the documentation (if it exists) and write small (but increasingly large) test programs until you become confident of your understanding of the software. I've found that, in the long run, this kind of self-training more than pays for itself in terms of saved time. Many of the programming examples in this book are constructed in ways that encourage this learning approach.

I've primarily been a software engineer and designer. However, I'm also a passionate teacher, and have spent several years teaching in both academic and commercial environments. I've run many week-long courses teaching UNIX system programming, and that experience informs the writing of this book.

I've been using Linux for about half as long as I've been using UNIX, and, over that time, my interest has increasingly centered on the boundary between the kernel and user space: the Linux programming interface. This interest has drawn me into a number of interrelated activities. I intermittently provide input and bug reports for the POSIX/SUS standard; I carry out tests and design reviews of new user-space interfaces added to the Linux kernel (and have helped find and fix many code and design bugs in those interfaces); I've been a regular speaker at conferences on topics related to interfaces and their documentation; and I've been invited on a number of occasions to the annual Linux Kernel Developers Summit. The common thread tying all of these activities together is my most visible contribution in the Linux world: my work on the *man-pages* project (*http://www.kernel.org/doc/man-pages/*).

The *man-pages* project provides pages in sections 2, 3, 4, 5, and 7 of the Linux manual pages. These are the manual pages describing the programming interfaces provided by the Linux kernel and the GNU C library—the same topic area as this book. I've been involved with *man-pages* for more than a decade. Since 2004, I've been the project maintainer, a task that involves, in roughly equal measure, writing documentation, reading kernel and library source code, and writing programs to verify the details for documentation. (Documenting an interface is a great way to find bugs in that interface.) I've also been the biggest contributor to *man-pages*—of the approximately 900 pages in *man-pages*, I am the author of 140, and the coauthor of another 125. So, even before you picked up this book, it's quite likely you've read some of my published work. I hope that you've found that work useful, and that you'll find this book even more so.

Acknowledgements

Without the support of a good many people, this book would have been far less than it is. It is a great pleasure to thank them.

A large team of technical reviewers around the world read drafts, found errors, pointed out confusing explanations, suggested rewordings and diagrams, tested programs, proposed exercises, identified aspects of the behavior of Linux and other UNIX implementations that I was not aware of, and offered support and encouragement. Many reviewers generously supplied insights and comments that I was able to incorporate into the book, at times making me look more knowledgeable than I am. Any mistakes that remain are, of course, my own.

Thanks especially to the following reviewers (listed alphabetically by surname), who either commented on large sections of the manuscript, commented extensively on smaller sections of the manuscript, or (magnificently) commented extensively on large sections of the manuscript:

- Christophe Blaess is a consulting software engineer and professional trainer who specializes in industrial (realtime and embedded) applications of Linux. Christophe is the author of *Programmation système en C sous Linux*, a fine French book covering many of the same topics as this book. He generously read and commented on many chapters of my book.
- David Butenhof (Hewlett-Packard) was a member of the original working group for POSIX threads and for the Single UNIX Specification threads extensions, and is the author of *Programming with POSIX Threads*. He wrote the original DCE Threads reference implementation for the Open Software Foundation,

and was lead architect of the threads implementation for OpenVMS and Digital UNIX. David reviewed the threads chapters, suggested many improvements, and patiently corrected several details of my understanding of the POSIX threads API.

- Geoff Clare works at The Open Group on their UNIX conformance test suites, has been involved with UNIX standardization for more than 20 years, and is one of half a dozen key participants in the Austin Group, which develops the joint standard that forms POSIX.1 and the base volumes of the Single UNIX Specification. Geoff provided detailed review of parts of the manuscript related to standard UNIX interfaces, patiently and politely suggested numerous fixes and improvements, spotted many obscure bugs, and provided much assistance in focusing on the importance of standards for portable programming.
- Loïc Domaigné (then at German Air Traffic Control) is a software systems engineer working on the design and development of distributed, concurrent, and fault-tolerant embedded systems with hard realtime requirements. He provided review input for the threads specification in SUSv3, and is an enthusiastic educator and knowledgeable contributor in various online technical forums. Loïc carried out a detailed review of the threads chapters, as well as many other parts of the book. He also implemented a number of clever programs to verify details of the Linux threads implementation, provided a great deal of enthusiasm and encouragement, and proposed numerous ideas to improve the overall presentation of the material.
- Gert Döring programmed *mgetty* and *sendfax*, a pair of programs that together are one of the most widely used open source fax packages for UNIX and Linux. These days, he works mainly on building and operating large IPv4-based and IPv6-based networks, a task that includes working with colleagues across Europe to define the operational policies that ensure the smooth operation of the infrastructure of the Internet. Gert provided extensive and useful feedback on the chapters covering terminals, login accounting, process groups, sessions, and job control.
- Wolfram Gloger is an IT consultant who has worked on a range of Free and Open Source Software (FOSS) projects in the past decade and a half. Among other things, Wolfram is the implementer of the *malloc* package used in the GNU C library. Currently, he works on web services development, with a particular focus on E-learning, although he still does occasional work on the kernel and system libraries. Wolfram reviewed a number of chapters, especially helping with my discussion of memory-related topics.
- Fernando Gont is a member of the Centro de Estudios de Informática (CEDI) at the Universidad Tecnológica Nacional, Argentina. He focuses on Internet engineering, with active participation in the Internet Engineering Task Force (IETF), where he has authored a number of *Request for Comments* (RFC) documents. Fernando also works on security assessment of communications protocols for the UK Centre for the Protection of National Infrastructure (CPNI), and has produced the first thorough security assessment of the TCP and IP protocols. Fernando provided a very thorough review of the network programming chapters, explained many details of TCP/IP, and suggested a multitude of improvements to the material.

- Andreas Grünbacher (SUSE Labs) is a kernel hacker and author of the Linux implementation of extended attributes and POSIX access control lists. Andreas provided thorough review of many chapters, much encouragement, and the single comment that probably most changed the structure of the book.
- Christoph Hellwig is a Linux storage and file-systems consultant and a well-known kernel hacker who has worked on many parts of the Linux kernel. Christoph kindly took time out from writing and reviewing Linux kernel patches to review several chapters of this book, suggesting many useful corrections and improvements.
- Andreas Jaeger led the development of the Linux port to the x86-64 architecture. As a GNU C Library developer, he ported the library to x86-64, and helped make the library standards-conformant in several areas, especially in the math library. He is currently Program Manager for openSUSE at Novell. Andreas reviewed far more chapters than I could possibly have hoped, suggested a multitude of improvements, and warmly encouraged the ongoing work on the book.
- Rick Jones, also known as "Mr. Netperf" (Networked Systems Performance Curmudgeon at Hewlett-Packard), provided valuable review of the network programming chapters.
- Andi Kleen (then at SUSE Labs) is a well-known and long-term kernel hacker who has worked on many and diverse areas of the Linux kernel, including networking, error handling, scalability, and low-level architecture code. Andi did an extensive review of the material on network programming, expanded my knowledge of many details of the Linux TCP/IP implementation, and suggested many ways to improve my presentation of the subject.
- Martin Landers (Google) was still a student when I had the good fortune to meet him as a colleague. Since then, he has managed to pack rather a lot into a short time, having worked variously as software architect, IT trainer, and professional hacker. I was fortunate indeed to have Martin as a reviewer. He contributed numerous incisive comments and corrections that greatly improved many chapters of the book.
- Jamie Lokier is a well-known kernel hacker who has been contributing to Linux development for 15 years. He nowadays describes himself as "a consultant in solving difficult problems that often have embedded Linux somewhere." Jamie provided an extraordinarily thorough review of the chapters on memory mappings, POSIX shared memory, and virtual memory operations. His comments corrected many details of my understanding of these topics and greatly improved the structure of the chapters.
- Barry Margolin has been a system programmer, system administrator, and support engineer throughout his 25-year career. He is currently a Senior Performance Engineer at Akamai Technologies. He is a frequent, well-respected contributor in various online forums discussing UNIX and Internet topics, and has reviewed a number of books on these topics. Barry reviewed a number of chapters of this book, suggesting many improvements.

- Paul Pluzhnikov (Google) was formerly the technical lead and a key developer of the *Insure++* memory-debugging tool. He is also a sometime *gdb* hacker, and a frequent responder in online forums answering questions on debugging, memory allocation, shared libraries, and run-time environments. Paul reviewed a wide range of chapters, suggesting many valuable improvements.
- John Reiser (with Tom London) carried out one of the earliest ports of UNIX to a 32-bit architecture: the VAX-11/780. He is also the creator of the *mmap()* system call. John reviewed many chapters (including, obviously, the chapter on *mmap()*), providing a multitude of historical insights and crystal-clear technical explanations that greatly improved the chapters.
- Anthony Robins (Associate Professor of Computer Science, University of Otago, New Zealand), a close friend of more than three decades, was the first reader of the drafts of several chapters, and offered valuable early comments and ongoing encouragement as the project evolved.
- Michael Schröder (Novell) is one of the main authors of the GNU *screen* program, a task that has imbued him with a thorough knowledge of the subtleties and differences in terminal-driver implementations. Michael reviewed the chapters covering terminals and pseudoterminals, and the chapter on process groups, sessions, and job control, providing much useful feedback.
- Manfred Spraul, who worked on the IPC code (among other things) in the Linux kernel, generously reviewed several of the chapters on IPC and suggested many improvements.
- Tom Swigg, a former UNIX training colleague at Digital, was an early reviewer who supplied important feedback on several chapters. A software engineer and IT trainer for more than 25 years, Tom currently works at London South Bank University, programming and supporting Linux in a VMware environment.
- Jens Thoms Törring is part of a fine tradition of physicists turned programmers, and has produced a variety of open source device drivers and other software. Jens read a surprisingly diverse collection of chapters, providing unique and valuable insight on how each could be improved.

Many other technical reviewers also read various parts of the book and made valuable comments. In alphabetical order by surname, thank you to George Anzinger (MontaVista Software), Stefan Becher, Krzysztof Benedyczak, Daniel Brahneborg, Andries Brouwer, Annabel Church, Dragan Cvetkovic, Floyd L. Davidson, Stuart Davidson (Hewlett-Packard Consulting), Kasper Dupont, Peter Fellinger (jambit GmbH), Mel Gorman (IBM), Niels Göllesch, Claus Gratzl, Serge Hallyn (IBM), Markus Hartinger (jambit GmbH), Richard Henderson (Red Hat), Andrew Josey (The Open Group), Dan Kegel (Google), Davide Libenzi, Robert Love (Google), H.J. Lu (Intel Corporation), Paul Marshall, Chris Mason, Michael Matz (SUSE), Trond Myklebust, James Peach, Mark Phillips (Automated Test Systems), Nick Piggin (SUSE Labs, Novell), Kay Johannes Potthoff, Florian Rampp, Stephen Rothwell (Linux Technology Centre, IBM), Markus Schwaiger, Stephen Tweedie (Red Hat), Britta Vargas, Chris Wright, Michal Wronski, and Umberto Zamuner.

Aside from technical review, I received many other kinds of help from various people and organizations.

Thanks to the following people for answering technical questions: Jan Kara, Dave Kleikamp, and Jon Snader. Thanks to Claus Gratzl and Paul Marshall for system management assistance.

Thanks to the Linux Foundation (LF), which, during 2008, funded me as a Fellow to work full time on the *man-pages* project and on testing and design review of the Linux programming interface. Although the Fellowship provided no direct financial support for working on this book, it did keep me and my family fed, and the ability to focus full time on documenting and testing the Linux programming interface was a boon to my "private" project. At a more individual level, thanks to Jim Zemlin for being my "interface" while working at the LF, and to the members of the LF Technical Advisory Board, who supported my application for the Fellowship.

Thanks to Alejandro Forero Cuervo for suggesting the title of the book!

More than 25 years ago, Robert Biddle intrigued me during my first degree with tales of UNIX, C, and Ratfor; thank you. Thanks to the following people, who, although not directly connected with this project, encouraged me on the path of writing during my second degree at the University of Canterbury, New Zealand: Michael Howard, Jonathan Mane-Wheoki, Ken Strongman, Garth Fletcher, Jim Pollard, and Brian Haig.

The late Richard Stevens wrote several superb books on UNIX programming and TCP/IP, which I, like a multitude of programmers, have found to be a wonderful source of technical information over the years. Readers of those books will note several visual aspects that are similar between my book and those of Richard Stevens. This is no accident. As I considered how to design my book, and looked around more generally at book designs, time and again, the approach employed by Richard Stevens seemed the best solution, and where this was so, I have employed the same visual approach.

Thanks to the following people and organizations for providing UNIX systems that enabled me to run test programs and verify details on other UNIX implementations: Anthony Robins and Cathy Chandra, for test systems at the University of Otago, New Zealand; Martin Landers, Ralf Ebner, and Klaus Tilk, for test systems at the Technische Universität in Munich, Germany; Hewlett-Packard, for making their *testdrive* systems freely available on the Internet; and Paul de Weerd for providing OpenBSD access.

Heartfelt thanks to two Munich companies, and their owners, who, in addition to providing me with flexible employment and enjoyable colleagues, were extraordinarily generous in allowing me to use their offices while writing this book. Thanks to Thomas Kahabka and Thomas Gmelch of exolution GmbH, and, especially, to Peter Fellinger and Markus Hartinger of jambit GmbH.

Thanks for various kinds of help to the following people: Dan Randow, Karen Korrel, Claudio Scalmazzi, Michael Schüpbach, and Liz Wright. Thanks to Rob Suisted and Lynley Cook for the photographs used on the front and back covers.

Thanks to the following people who encouraged and supported me in various ways on this project: Deborah Church, Doris Church, and Annie Currie.

Thanks to the team at No Starch Press for all sorts of help on an enormous project. Thanks to Bill Pollock for being straight-talking from the start, having rock-solid faith in the project, and patiently keeping an eye on the project. Thanks to my initial production editor, Megan Dunchak. Thanks to my copyeditor, Marilyn Smith, who, despite my best efforts at clarity and consistency, still found many things to fix. Riley Hoffman had overall responsibility for layout and design of the book, and also took up the reins as production editor as we came into the home straight. Riley graciously bore with my many requests to achieve the right layout and produced a superb final result. Thank you.

I now know the truth of the cliché that a writer's family also pays the price of the writer's work. Thanks to Britta and Cecilia for their support, and for putting up with the many hours that I had to be away from family as I finished the book.

Permissions

Web site and source code of example programs

You can find further information about this book, including errata and source code for the example programs, at *http://www.man7.org/tlpi*.

Feedback

I welcome bug reports, suggestions for code improvements, and fixes to further improve code portability. Book bugs and general suggestions about how the explanations in the book can be improved are also welcome. Since changes in the Linux programming interface are varied and sometimes too frequent for one person to keep up with, I would be happy to receive suggestions about new and changed features that should be covered in a future edition of this book.

Michael Timothy Kerrisk
Munich, Germany and Christchurch, New Zealand
August 2010

mtk@man7.org