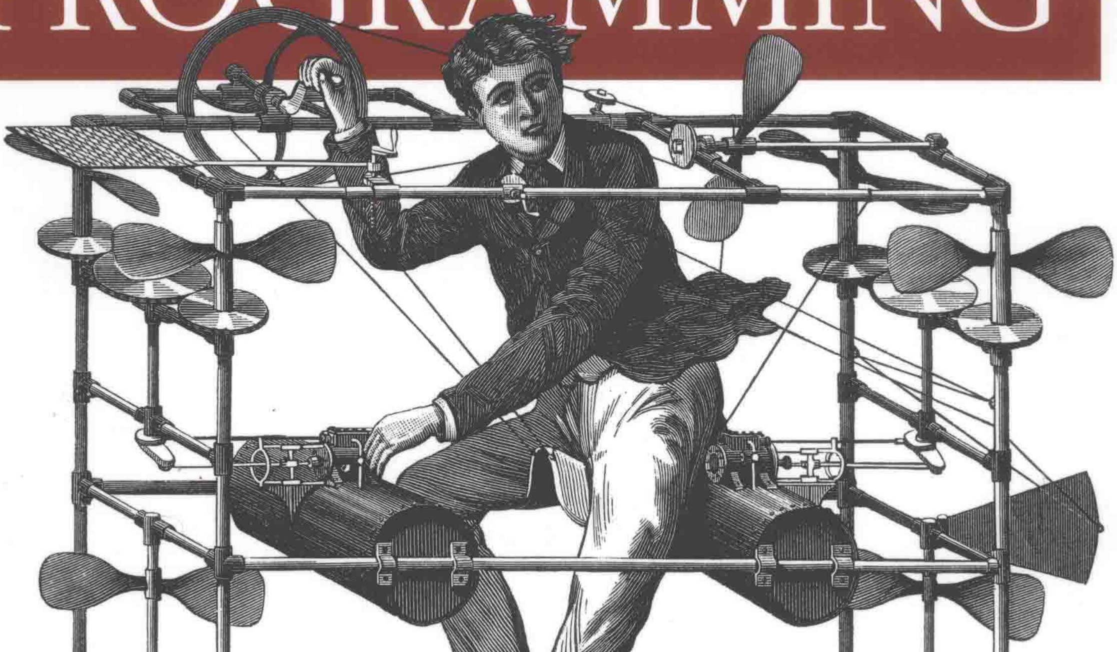
Linux系统编程（影印版）

第二版
基于Linux 3.0内核的修订

LINUX SYSTEM PROGRAMMING



O'REILLY®

東南大學出版社

ROBERT LOVE 著

第二版

Linux系统编程（影印版）

Linux System Programming

Robert Love 著

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

O'Reilly Media, Inc.授权东南大学出版社出版

南京 东南大学出版社

图书在版编目（CIP）数据

Linux 系统编程：第 2 版：英文 /(美)洛夫（Love, R.）著 . —影印本 . —南京：东南大学出版社，2014.1

书名原文：Linux System Programming, 2E

ISBN 978-7-5641-4601-6

I. ① L… II. ①洛… III. ① Linux 操作系统－程序设计－英文 IV. ① TP316.89

中国版本图书馆 CIP 数据核字（2013）第 246058 号

江苏省版权局著作权合同登记

图字：10-2013-371 号

Linux 系统编程 第二版（影印版）

出版发行：	东南大学出版社
地　　址：	南京四牌楼 2 号　　邮编：210096
出 版 人：	江建中
网　　址：	http://www.seupress.com
电子邮件：	press@seupress.com
印　　刷：	扬中市印刷有限公司
开　　本：	787 毫米 × 980 毫米　16 开本
印　　张：	28.5
字　　数：	558 千字
版　　次：	2014 年 1 月第 1 版
印　　次：	2014 年 1 月第 1 次印刷
书　　号：	ISBN 978-7-5641-4601-6
定　　价：	78.00 元（册）

本社图书若有印装质量问题，请直接与营销部联系。电话（传真）：025-83791830

For Doris and Helen.

Foreword

There is an old line that Linux kernel developers like to throw out when they are feeling grumpy: “User space is just a test load for the kernel.”

By muttering this line, the kernel developers aim to wash their hands of all responsibility for any failure to run user-space code as well as possible. As far as they're concerned, user-space developers should just go away and fix their own code, as any problems are definitely not the kernel's fault.

To prove that it usually is not the kernel that is at fault, one leading Linux kernel developer has been giving a “Why User Space Sucks” talk to packed conference rooms for more than three years now, pointing out real examples of horrible user-space code that everyone relies on every day. Other kernel developers have created tools that show how badly user-space programs are abusing the hardware and draining the batteries of unsuspecting laptops.

But while user-space code might be just a “test load” for kernel developers to scoff at, it turns out that all of these kernel developers also depend on that user-space code every day. If it weren't present, all the kernel would be good for would be to print out alternating ABABAB patterns on the screen.

Right now, Linux is the most flexible and powerful operating system that has ever been created, running everything from the tiniest cell phones and embedded devices to more than 90 percent of the world's top 500 supercomputers. No other operating system has ever been able to scale so well and meet the challenges of all of these different hardware types and environments.

And along with the kernel, code running in user space on Linux can also operate on all of those platforms, providing the world with real applications and utilities people rely on.

In this book, Robert Love has taken on the unenviable task of teaching the reader about almost every system call on a Linux system. In so doing, he has produced a tome that will allow you to fully understand how the Linux kernel works from a user-space perspective, and also how to harness the power of this system.

The information in this book will show you how to create code that will run on all of the different Linux distributions and hardware types. It will allow you to understand how Linux works and how to take advantage of its flexibility.

In the end, this book teaches you how to write code that doesn't suck, which is the best thing of all.

—Greg Kroah-Hartman

Preface

This book is about system programming on Linux. *System programming* is the practice of writing *system software*, which is code that lives at a low level, talking directly to the kernel and core system libraries. Put another way, the topic of the book is Linux system calls and low-level functions such as those defined by the C library.

While many books cover system programming for Unix systems, few tackle the subject with a focus solely on Linux, and fewer still address the very latest Linux releases and advanced Linux-only interfaces. Moreover, this book benefits from a special touch: I have written a lot of code for Linux, both for the kernel and for system software built thereon. In fact, I have implemented some of the system calls and other features covered in this book. Consequently, this book carries a lot of insider knowledge, covering not just how the system interfaces *should* work, but how they *actually* work and how you can use them most efficiently. This book, therefore, combines in a single work a tutorial on Linux system programming, a reference manual covering the Linux system calls, and an insider's guide to writing smarter, faster code. The text is fun and accessible, and regardless of whether you code at the system level on a daily basis, this book will teach you tricks that will enable you to be a better software engineer.

Audience and Assumptions

The following pages assume that the reader is familiar with C programming and the Linux programming environment—not necessarily well-versed in the subjects, but at least acquainted with them. If you are not comfortable with a Unix text editor—Emacs and *vim* being the most common and highly regarded—start playing with one. You'll also want to be familiar with the basics of using *gcc*, *gdb*, *make*, and so on. Plenty of other books on tools and practices for Linux programming are out there; Appendix B at the end of this book lists several useful references.

I've made few assumptions about the reader's knowledge of Unix or Linux system programming. This book will start from the ground up, beginning with the basics, and winding its way up to the most advanced interfaces and optimization tricks. Readers of all levels, I hope, will find this work worthwhile and learn something new. In the course of writing the book, I certainly did.

Similarly, I make few assumptions about the persuasion or motivation of the reader. Engineers wishing to program (better) at the system level are obviously targeted, but higher-level programmers looking for a stronger foundation will also find a lot to interest them. Merely curious hackers are also welcome, for this book should satiate that hunger, too. This book aims to cast a net wide enough to satisfy most programmers.

Regardless of your motives, above all else, *have fun*.

Contents of This Book

This book is broken into 11 chapters and two appendices.

Chapter 1, Introduction and Essential Concepts
: This chapter serves as an introduction, providing an overview of Linux, system programming, the kernel, the C library, and the C compiler. Even advanced users should visit this chapter.

Chapter 2, File I/O
: This chapter introduces files, the most important abstraction in the Unix environment, and file I/O, the basis of the Linux programming mode. It covers reading from and writing to files, along with other basic file I/O operations. The chapter culminates with a discussion on how the Linux kernel implements and manages files.

Chapter 3, Buffered I/O
: This chapter discusses an issue with the basic file I/O interfaces—buffer size management—and introduces buffered I/O in general, and standard I/O in particular, as solutions.

Chapter 4, Advanced File I/O
: This chapter completes the I/O troika with a treatment on advanced I/O interfaces, memory mappings, and optimization techniques. The chapter is capped with a discussion on avoiding seeks and the role of the Linux kernel's I/O scheduler.

Chapter 5, Process Management
: This chapter introduces Unix's second most important abstraction, the *process*, and the family of system calls for basic process management, including the venerable *fork*.

Chapter 6, Advanced Process Management
: This chapter continues the treatment with a discussion of advanced process management, including real-time processes.

Chapter 7, Threading
: This chapter discusses threads and multithreaded programming. It focuses on higher-level design concepts. It includes an introduction to the POSIX threading API, known as Pthreads.

Chapter 8, File and Directory Management
: This chapter discusses creating, moving, copying, deleting, and otherwise managing files and directories.

Chapter 9, Memory Management
: This chapter covers memory management. It begins by introducing Unix concepts of memory, such as the process address space and the page, and continues with a discussion of the interfaces for obtaining memory from and returning memory to the kernel. The chapter concludes with a treatment on advanced memory-related interfaces.

Chapter 10, Signals
: This chapter covers signals. It begins with a discussion of signals and their role on a Unix system. It then covers signal interfaces, starting with the basic and concluding with the advanced.

Chapter 11, Time
: This chapter discusses time, sleeping, and clock management. It covers the basic interfaces up through POSIX clocks and high-resolution timers.

Appendix A
: The first appendix reviews many of the language extensions provided by *gcc* and GNU C, such as attributes for marking a function constant, pure, or inline.

Appendix B
: This bibliography of recommended reading lists both useful supplements to this work, and books that address prerequisite topics not covered herein.

Versions Covered in This Book

The Linux system interface is definable as the application binary interface and application programming interface provided by the triplet of the Linux kernel (the heart of the operating system), the GNU C library (*glibc*), and the GNU C Compiler (*gcc*—now formally called the GNU Compiler Collection, but we are concerned only with C). This book covers the system interface defined by Linux kernel version 3.9, *glibc* version 2.17, and *gcc* version 4.8. Interfaces in this book should be forward compatible with newer versions of the kernel, *glibc*, and *gcc*. That is, newer versions of these components should continue to obey the interfaces and behavior documented in this book. Similarly, many of the interfaces discussed in this book have long been part of Linux and are thus backward compatible with older versions of the kernel, *glibc*, and *gcc*.

If any evolving operating system is a moving target, Linux is a rabid cheetah. Progress is measured in days, not years, and frequent releases of the kernel and other components constantly morph the playing field. No book can hope to capture such a dynamic beast in a timeless fashion.

Nonetheless, the programming environment defined by system programming is *set in stone*. Kernel developers go to great pains not to break system calls, the *glibc* developers highly value forward *and* backward compatibility, and the Linux toolchain generates compatible code across versions. Consequently, while Linux may be constantly on the go, Linux system programming remains stable, and a book based on a snapshot of the system, especially at this point in Linux's lifetime, has immense staying power. What I am trying to say is simple: don't worry about system interfaces changing and *buy this book*!

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.

This icon signifies a warning or caution.

Most of the code in this book is in the form of brief, but reusable, code snippets. They look like this:

```
while (1) {
        int ret;

        ret = fork ();
        if (ret == -1)
                perror ("fork");
}
```

Great pains have been taken to provide code snippets that are concise but usable. No special header files, full of crazy macros and illegible shortcuts, are required. Instead of building a few gigantic programs, this book is filled with many simple examples. As the examples are descriptive and fully usable, yet small and clear, I hope they will provide a useful tutorial on the first read, and remain a good reference on subsequent passes.

Nearly all the examples in this book are self-contained. This means you can easily copy them into your text editor and put them to use. Unless otherwise mentioned, all the code snippets should build without any special compiler flags. (In a few cases, you need to link with a special library.) I recommend the following command to compile a source file:

```
$ gcc -Wall -Wextra -O2 -g -o snippet snippet.c
```

This compiles the source file *snippet.c* into the executable binary *snippet*, enabling many warning checks, significant but sane optimizations, and debugging. The code in this book should compile using this command without errors or warnings—although of course, you might have to build a skeleton program around the snippet first.

When a section introduces a new function, it is in the usual Unix manpage format, which looks like this:

```
#include <fcntl.h>

int posix_fadvise (int fd, off_t pos, off_t len, int advice);
```

The required headers, and any needed definitions, are at the top, followed by a full prototype of the call.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Linux System Programming,* Second Edition, by Robert Love (O'Reilly). Copyright 2013 Robert Love, 978-1-449-33953-1."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Because the snippets in this book are numerous but short, they are not available in an online repository.

Safari® Books Online

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://oreil.ly/Linux_Sys_2E*.

To comment or ask technical questions about this book, send email to *bookques tions@oreilly.com.*

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com.*

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Many hearts and minds contributed to the completion of this manuscript. While no list would be complete, it is my sincere pleasure to acknowledge the assistance and friendship of individuals who provided encouragement, knowledge, and support along the way.

Andy Oram is a phenomenal editor and human being. This effort would have been impossible without his hard work. A rare breed, Andy couples deep technical knowledge with a poetic command of the English language.

This book was blessed with phenomenal technical reviewers, true masters of their craft, without whom this work would pale in comparison to the final product you now read. The technical reviewers were Jeremy Allison, Robert P. J. Day, Kenneth Geisshirt, Joey Shaw, and James Willcox. Despite their toils, any errors remain my own.

My colleagues at Google remain the smartest, most dedicated group of engineers with whom I have had the pleasure to work. Each day is a challenge in the best use of that word. Thank you for the system-level projects that helped shape this text and an atmosphere that encourages pursuits such as this work.

For numerous reasons, thanks and respect to Paul Amici, Mikey Babbitt, Nat Friedman, Miguel de Icaza, Greg Kroah-Hartman, Doris Love, Linda Love, Tim O'Reilly, Salvatore Ribaudo and family, Chris Rivera, Carolyn Rodon, Joey Shaw, Sarah Stewart, Peter Teichman, Linus Torvalds, Jon Trowbridge, Jeremy VanDoren and family, Luis Villa, Steve Weisberg and family, and Helen Whisnant.

Final thanks to my parents, Bob and Elaine.

—Robert Love, Boston

Table of Contents