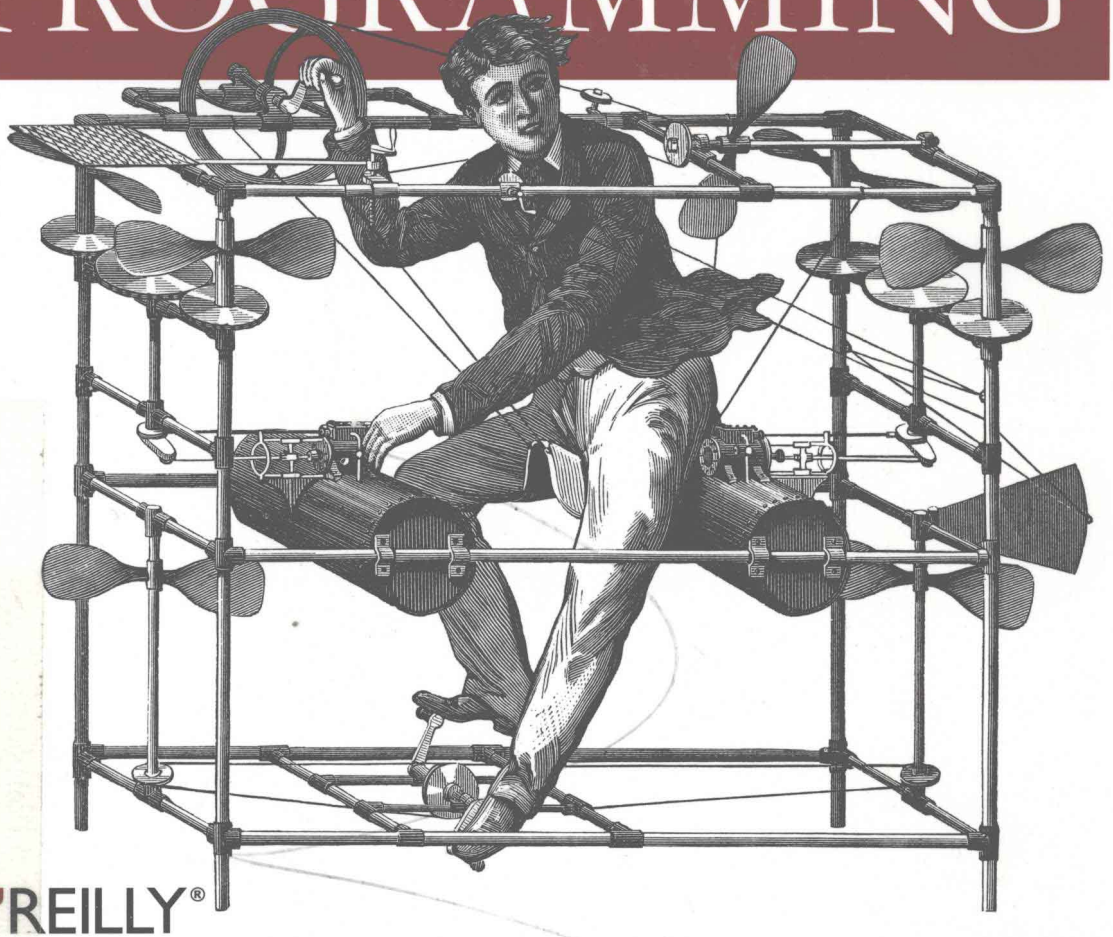
LINUX系统编程（影印版）

# LINUX SYSTEM PROGRAMMING



O'REILLY®
東南大學出版社

ROBERT LOVE 著

# LINUX
## System Programming

## LINUX系统编程(影印版)



*Robert Love*

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo
O'Reilly Media, Inc.授权东南大学出版社出版
东南大学出版社

**图书在版编目（CIP）数据**

Linux 系统编程：英文 /（美）洛夫（Love, R.）著.
影印本 . — 南京：东南大学出版社，2008.3
书名原文：Linux System Programming
ISBN 978-7-5641-1141-0

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

中国版本图书馆 CIP 数据核字（2008）第 024164 号

江苏省版权局著作权合同登记
图字：10-2007-225 号



Linux 系统编程

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　汉
网　　址：http://press.seu.edu.cn
电子邮件：press@seu.edu.cn
印　　刷：扬中市印刷有限公司
开　　本：787 毫米 × 980 毫米　16 开本
印　　张：24.5 印张
字　　数：412 千字
版　　次：2008 年 3 月第 1 版
印　　次：2008 年 3 月第 1 次印刷
书　　号：ISBN 978-7-5641-1141-0/TP · 185
印　　数：1~2500 册
定　　价：59.00 元（册）

本社图书若有印装质量问题，请直接与读者服务部联系。电话（传真）：025-83792328

# O'Reilly Media, Inc. 介绍

为了满足读者对网络和软件技术知识的迫切需求，世界著名计算机图书出版机构O'Reilly Media, Inc. 授权东南大学出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly Media, Inc. 是世界上在 Unix、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时也是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》（被纽约公共图书馆评为二十世纪最重要的50本书之一）到GNN（最早的Internet门户和商业网站），再到WebSite（第一个桌面PC的Web服务器软件），O'Reilly Media, Inc. 一直处于Internet发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc. 是最稳定的计算机图书出版商——每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc. 具有深厚的计算机专业背景，这使得O'Reilly Media, Inc. 形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc. 所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc. 还有许多固定的作者群体——他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc. 依靠他们及时地推出图书。因为O'Reilly Media, Inc. 紧密地与计算机业界联系着，所以O'Reilly Media, Inc. 知道市场上真正需要什么图书。

# 出版说明

随着计算机技术的成熟和广泛应用，人类正在步入一个技术迅猛发展的新时期。计算机技术的发展给人们的工业生产、商业活动和日常生活都带来了巨大的影响。然而，计算机领域的技术更新速度之快也是众所周知的，为了帮助国内技术人员在第一时间了解国外最新的技术，东南大学出版社和美国 O'Reilly Meida, Inc.达成协议，将陆续引进该公司的代表前沿技术或者在某专项领域享有盛名的著作，以影印版或者简体中文版的形式呈献给读者。其中，影印版书籍力求与国外图书“同步”出版，并且“原汁原味”展现给读者。

我们真诚地希望，所引进的书籍能对国内相关行业的技术人员、科研机构的研究人员和高校师生的学习和工作有所帮助，对国内计算机技术的发展有所促进。也衷心期望读者提出宝贵的意见和建议。

最新出版的影印版图书，包括：

- 深入浅出 SQL（影印版）
- JavaScript & DHTML Cookbook 第二版（影印版）
- 集体智慧编程（影印版）
- SOA 实践（影印版）
- 学习 PHP & MySQL 第二版（影印版）
- Linux 系统编程（影印版）
- Beautiful Code（影印版）
- Mac OS X: The Missing Manual, Leopard Editon（影印版）
- 高性能网站（影印版）
- WPF 编程 第二版（影印版）
- 敏捷开发艺术（影印版）
- 学习 Python 第三版（影印版）
- Ruby 程序设计语言（影印版）

# 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 70 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—specifically, 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 other 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 (if any) 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 (the programmer) 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 write better code.

## 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 have not yet read any books on the C programming language, such as the classic Brian W. Kernighan and Dennis M. Ritchie work *The C Programming Language* (Prentice Hall; the book is familiarly known as K&R), I highly recommend you check one out. 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; the bibliography 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.

Nor do I make assumptions about the persuasion or motivation of the reader. Engineers wishing to program (better) at a low level are obviously targeted, but higher-level programmers looking for a stronger standing on the foundations on which they rest will also find a lot to interest them. Simply curious hackers are also welcome, for this book should satiate their hunger, too. Whatever readers want and need, this book should cast a net wide enough—as least as far as Linux system programming is concerned—to satisfy them.

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

## Contents of This Book

This book is broken into 10 chapters, an appendix, and a bibliography.

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—trust me.

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

Chapter 8, *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 9, *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 10, *Time*
: This chapter discusses time, sleeping, and clock management. It covers the basic interfaces up through POSIX clocks and high-resolution timers.

Appendix, *GCC Extensions to the C Language*
: The Appendix reviews many of the optimizations provided by *gcc* and GNU C, such as attributes for marking a function constant, pure, and inline.

The book concludes with a bibliography of recommended reading, listing 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 2.6.22, *glibc* version 2.5, and *gcc* version 4.2. Interfaces in this book should be backward compatible with older versions (excluding new interfaces), and forward compatible to newer versions.

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 (particularly for the C language). 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 development, 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*
: Used for emphasis, new terms, URLs, foreign phrases, Unix commands and utilities, filenames, directory names, and pathnames.

`Constant width`
: Indicates header files, variables, attributes, functions, types, parameters, objects, macros, and other programming constructs.

*`Constant width italic`*
: Indicates text (for example, a pathname component) to be replaced with a user-supplied value.

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

Most of the code in this book is in the form of brief, but usable, 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 of the examples in this book are self-contained. This means you can easily copy them into your text editor, and put them to actual use. Unless otherwise mentioned, all of 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 with a special emphasized font, 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.

## Safari® Books Online


When you see a Safari® Books Online icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safari.oreilly.com*.

## 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 are 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 attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Linux System Programming* by Robert Love. Copyright 2007 O'Reilly Media, Inc., 978-0-596-00958-8."

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

## 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 this address:

*http://www.oreilly.com/catalog/9780596009588/*

To comment or ask technical questions about this book, you can send an email to the following address:

*bookquestions@oreilly.com*

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at this address:

*http://www.oreilly.com*

## 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.

Brian Jepson served brilliantly as editor for a period, and his sterling efforts continue to reverberate throughout this work as well.

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 Robert Day, Jim Lieb, Chris Rivera, Joey Shaw, and Alain Williams. Despite their toils, any errors remain my own.

Rachel Head performed flawlessly as copyeditor. In her aftermath, red ink decorated my written word—readers will certainly appreciate her corrections.

For numerous reasons, thanks and respect to Paul Amici, Mikey Babbitt, Keith Barbag, Jacob Berkman, Dave Camp, Chris DiBona, Larry Ewing, Nat Friedman, Albert Gator, Dustin Hall, Joyce Hawkins, Miguel de Icaza, Jimmy Krehl, Greg Kroah-Hartman, Doris Love, Jonathan Love, Linda Love, Tim O'Reilly, Aaron Matthews, John McCain, Randy O'Dowd, Salvatore Ribaudo and family, Chris Rivera, 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*

## About the Author

**Robert Love** has been a Linux user and hacker since the early days. He is active in—and passionate about—the Linux kernel and GNOME desktop communities. His recent contributions to the Linux kernel include work on the kernel event layer and inotify. GNOME-related contributions include Beagle, GNOME Volume Manager, NetworkManager, and Project Utopia. Currently, Robert works in the Open Source Program Office at Google.

As an author, Robert is responsible for *Linux Kernel Development* (Novell Press), now in its second edition. He is also a coauthor of the fifth edition of O'Reilly's *Linux in a Nutshell*. A contributing editor for *Linux Journal*, Robert has written many articles and has been invited to speak around the world on Linux.

Robert graduated from the University of Florida with a B.A. in mathematics and a B.S. in computer science. Hailing from south Florida, he now calls Boston home.

## Colophon

The image on the cover of *Linux System Programming* is a man in a flying machine. Well before the Wright brothers achieved their first controlled heavier-than-air flight in 1903, people around the world attempted to fly by simple and elaborate machines. In the second or third century, Zhuge Liang of China reportedly flew in a Kongming lantern, the first hot air balloon. Around the fifth or sixth centuries, many Chinese people purportedly attached themselves to large kites to fly through the air.

It is also said that the Chinese created spinning toys that were early versions of helicopters, the designs of which may have inspired Leonardo da Vinci in his initial attempts at a solution to human flight. da Vinci also studied birds and designed parachutes, and in 1845, he designed an ornithopter, a wing-flapping machine meant to carry humans through the air. Though he never built it, the ornithopter's birdlike structure influenced the design of flying machines throughout the centuries.

The flying machine depicted on the cover is more elaborate than James Means' model soaring machine of 1893, which had no propellers. Means later printed an instruction manual for his soaring machine, which in part states that "the summit of Mt. Willard, near the Crawford House, N.H., will be found an excellent place" to experiment with the machines.

But such experimentation was often dangerous. In the late nineteenth century, Otto Lilienthal built monoplanes, biplanes, and gliders. He was the first to show that control of human flight was within reach, and he gained the nickname "father of aerial testing," as he conducted more than 2,000 glider flights, sometimes traveling more than a thousand feet. He died in 1896 after breaking his spine during a crash landing.

Flying machines are also known as mechanical birds and airships, and are occasionally called by more colorful names such as the Artificial Albatross. Enthusiasm for flying machines remains high, as aeronautical buffs still build early flying machines today.

The cover image and chapter opening graphics are from the Dover Pictorial Archive. The cover font is Adobe ITC Garamond. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed.

# Table of Contents