

# Advanced Linux Programming

## Contents At a Glance

- I** Advanced UNIX Programming with Linux
  - 1** Getting Started **3**
  - 2** Writing Good GNU/Linux Software **17**
  - 3** Processes **45**
  - 4** Threads **61**
  - 5** Interprocess Communication **95**
- II** Mastering Linux
  - 6** Devices **129**
  - 7** The */proc* File System **147**
  - 8** Linux System Calls **167**
  - 9** Inline Assembly Code **189**
  - 10** Security **197**
  - 11** A Sample GNU/Linux Application **219**
- III** Appendixes
  - A** Other Development Tools **259**
  - B** Low-Level I/O **281**
  - C** Table of Signals **301**
  - D** Online Resources **303**
  - E** Open Publication License Version 1.0 **305**
  - F** GNU General Public License **309**



# Advanced Linux Programming

---

Mark Mitchell, Jeffrey Oldham,  
and Alex Samuel



[www.newriders.com](http://www.newriders.com)

201 West 103rd Street, Indianapolis, Indiana 46290

An Imprint of Pearson Education

Boston • Indianapolis • London • Munich • New York • San Francisco

# **Advanced Linux Programming**

**Copyright © 2001 by New Riders Publishing**

FIRST EDITION: June, 2001

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the publisher, except for the inclusion of brief quotations in a review.

International Standard Book Number: 0-7357-1043-0

Library of Congress Catalog Card Number: 00-105343

05 04 03 02 01 7 6 5 4 3 2 1

Interpretation of the printing code: The rightmost double-digit number is the year of the book's printing; the rightmost single-digit number is the number of the book's printing. For example, the printing code 01-1 shows that the first printing of the book occurred in 2001.

Composed in Bembo and MCPdigital by New Riders Publishing.

Printed in the United States of America.

## **Trademarks**

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. New Riders Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

PostScript is a trademark of Adobe Systems, Inc.

Linux is a trademark of Linus Torvalds.

## **Warning and Disclaimer**

This book is designed to provide information about *Advanced Linux Programming*. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied.

The information is provided on an as-is basis. The authors and New Riders Publishing shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the discs or programs that may accompany it.

### **Publisher**

David Dwyer

### **Associate Publisher**

Al Valvano

### **Executive Editor**

Stephanie Wall

### **Managing Editor**

Gina Brown

### **Acquisitions Editor**

Ann Quinn

### **Development Editor**

Laura Loveall

### **Product Marketing Manager**

Stephanie Layton

### **Publicity Manager**

Susan Petro

### **Project Editor**

Caroline Wise

### **Copy Editor**

Krista Hansing

### **Senior Indexer**

Cheryl Lenser

### **Manufacturing Coordinator**

Jim Conway

### **Book Designer**

Louisa Klucznik

### **Cover Designer**

Brainstorm Design, Inc.

### **Cover Production**

Aren Howell

### **Proofreader**

Debra Neel

### **Composition**

Amy Parker





# Table of Contents

## **I Advanced UNIX Programming with Linux 1**

### **1 Getting Started 3**

- 1.1 Editing with Emacs 4
- 1.2 Compiling with GCC 6
- 1.3 Automating the Process with GNU Make 9
- 1.4 Debugging with GNU Debugger (GDB) 11
- 1.5 Finding More Information 13

### **2 Writing Good GNU/Linux Software 17**

- 2.1 Interaction With the Execution Environment 17
- 2.2 Coding Defensively 30
- 2.3 Writing and Using Libraries 36

### **3 Processes 45**

- 3.1 Looking at Processes 45
- 3.2 Creating Processes 48
- 3.3 Signals 52
- 3.4 Process Termination 55

### **4 Threads 61**

- 4.1 Thread Creation 62
- 4.2 Thread Cancellation 69
- 4.3 Thread-Specific Data 72
- 4.4 Synchronization and Critical Sections 77
- 4.5 GNU/Linux Thread Implementation 92
- 4.6 Processes Vs. Threads 94

**5 Interprocess Communication 95**

- 5.1 Shared Memory 96
- 5.2 Processes Semaphores 101
- 5.3 Mapped Memory 105
- 5.4 Pipes 110
- 5.5 Sockets 116

**II Mastering Linux 127****6 Devices 129**

- 6.1 Device Types 130
- 6.2 Device Numbers 130
- 6.3 Device Entries 131
- 6.4 Hardware Devices 133
- 6.5 Special Devices 136
- 6.6 PTYs 142
- 6.7 *ioctl* 144

**7 The /proc File System 147**

- 7.1 Extracting Information from */proc* 148
- 7.2 Process Entries 150
- 7.3 Hardware Information 158
- 7.4 Kernel Information 160
- 7.5 Drives, Mounts, and File Systems 161
- 7.6 System Statistics 165

**8 Linux System Calls 167**

- 8.1 Using *strace* 168
- 8.2 *access*: Testing File Permissions 169
- 8.3 *fcntl*: Locks and Other File Operations 171
- 8.4 *fsync* and *fdatasync*: Flushing Disk Buffers 173
- 8.5 *getrlimit* and *setrlimit*: Resource Limits 174
- 8.6 *getrusage*: Process Statistics 175
- 8.7 *gettimeofday*: Wall-Clock Time 176

- 8.8 The *mlock* Family: Locking Physical Memory 177
- 8.9 *mprotect*: Setting Memory Permissions 179
- 8.10 *nanosleep*: High-Precision Sleeping 181
- 8.11 *readlink*: Reading Symbolic Links 182
- 8.12 *sendfile*: Fast Data Transfers 183
- 8.13 *setitimer*: Setting Interval Timers 185
- 8.14 *sysinfo*: Obtaining System Statistics 186
- 8.15 *uname* 187

## **9 Inline Assembly Code 189**

- 9.1 When to Use Assembly Code 190
- 9.2 Simple Inline Assembly 191
- 9.3 Extended Assembly Syntax 192
- 9.4 Example 194
- 9.5 Optimization Issues 196
- 9.6 Maintenance and Portability Issues 196

## **10 Security 197**

- 10.1 Users and Groups 198
- 10.2 Process User IDs and Process Group IDs 199
- 10.3 File System Permissions 200
- 10.4 Real and Effective IDs 205
- 10.5 Authenticating Users 208
- 10.6 More Security Holes 211

## **11 A Sample GNU/Linux Application 219**

- 11.1 Overview 219
- 11.2 Implementation 221
- 11.3 Modules 239
- 11.4 Using the Server 252
- 11.5 Finishing Up 255

### **III Appendixes 257**

#### **A Other Development Tools 259**

- A.1 Static Program Analysis 259
- A.2 Finding Dynamic Memory Errors 261
- A.3 Profiling 269

#### **B Low-Level I/O 281**

- B.1 Reading and Writing Data 282
- B.2 *stat* 291
- B.3 Vector Reads and Writes 293
- B.4 Relation to Standard C Library I/O Functions 295
- B.5 Other File Operations 296
- B.6 Reading Directory Contents 296

#### **C Table of Signals 301**

#### **D Online Resources 303**

- D.1 General Information 303
- D.2 Information About GNU/Linux Software 304
- D.3 Other Sites 304

#### **E Open Publication License Version 1.0 305**

- I. Requirement on Both Unmodified and Modified Versions 305
- II. Copyright 306
- III. Scope of License 306
- IV. Requirements on Modified Works 306
- V. Good-Practice Recommendations 306
- VI. License Options 307
- Open Publication Policy Appendix 307

**F GNU General Public License 309**

Preamble 309

Terms and Conditions for Copying,  
Distribution and Modification 310

End of Terms and Conditions 315

How to Apply These Terms to Your New  
Programs 315

**Index 317**



## Table of Program Listings

- 1.1 main.c (C source file), 6
- 1.2 reciprocal.cpp (C++ source file), 6
- 1.3 reciprocal.hpp (header file), 7
- 2.1 arglist.c (argc and argv parameters), 18
- 2.2 getopt\_long.c (getopt\_long function), 21
- 2.3 print\_env.c (printing execution environment), 26
- 2.4 client.c (network client program), 26
- 2.5 temp\_file.c (mkstemp function), 28
- 2.6 readfile.c (resource allocation during error checking), 35
- 2.7 test.c (library contents), 37
- 2.8 app.c (program with library functions), 37
- 2.9 tifftest.c (libtiff library), 40
- 3.1 print-pid.c (printing process IDs), 46
- 3.2 system.c (system function), 48
- 3.3 fork.c (fork function), 49
- 3.4 fork-exec.c (fork and exec functions), 51
- 3.5 sigusr1.c (signal handlers), 54
- 3.6 zombie.c (zombie processes), 58
- 3.7 sigchld.c (cleaning up child processes), 60
- 4.1 thread-create.c (creating threads), 63
- 4.2 thread-create2 (creating two threads), 64
- 4.3 thread-create2.c (revised main function), 65
- 4.4 primes.c (prime number computation in a thread), 67
- 4.5 detached.c (creating detached threads), 69
- 4.6 critical-section.c (critical sections), 71
- 4.7 tsd.c (thread-specific data), 73
- 4.8 cleanup.c (cleanup handlers), 75
- 4.9 cxx-exit.cpp (C++ thread cleanup), 76
- 4.10 job-queue1.c (thread race conditions), 78
- 4.11 job-queue2.c (mutexes), 80
- 4.12 job-queue3.c (semaphores), 84
- 4.13 spin-condvar.c (condition variables), 87

- 4.14 condvar.c (condition variables), 90
- 4.15 thread-pid (printing thread process IDs), 92
- 5.1 shm.c (shared memory), 99
- 5.2 sem\_all\_deall.c (semaphore allocation and deallocation), 102
- 5.3 sem\_init.c (semaphore initialization), 102
- 5.4 sem\_pv.c (semaphore wait and post operations), 104
- 5.5 mmap-write.c (mapped memory), 106
- 5.6 mmap-read.c (mapped memory), 107
- 5.7 pipe.c (parent-child process communication), 111
- 5.8 dup2.c (output redirection), 113
- 5.9 popen.c (popen command), 114
- 5.10 socket-server.c (local sockets), 120
- 5.11 socket-client.c (local sockets), 121
- 5.12 socket-inet.c (Internet-domain sockets), 124
- 6.1 random\_number.c (random number generation), 138
- 6.2 cdrom-eject.c (ioctl example), 144
- 7.1 clock-speed.c (cpu clock speed from /proc/cpuinfo), 149
- 7.2 get-pid.c (process ID from /proc/self), 151
- 7.3 print-arg-list.c (printing process argument lists), 153
- 7.4 print-environment.c (process environment), 154
- 7.5 get-exe-path.c (program executable path), 155
- 7.6 open-and-spin.c (opening files), 157
- 7.7 print-uptime.c (system uptime and idle time), 165
- 8.1 check-access.c (file access permissions), 170
- 8.2 lock-file.c (write locks), 171
- 8.3 write\_journal\_entry.c (data buffer flushing), 173
- 8.4 limit-cpu.c (resource limits), 175
- 8.5 print-cpu-times.c (process statistics), 176

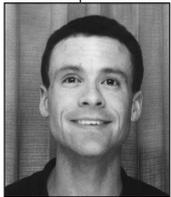
- 8.6 print-time.c (date/time printing), 177
- 8.7 mprotect.c (memory access), 180
- 8.8 better\_sleep.c (high-precision sleep), 182
- 8.9 print-symlink.c (symbolic links), 183
- 8.10 copy.c (sendfile system call), 184
- 8.11 itimer.c (interval timers), 185
- 8.12 sysinfo.c (system statistics), 187
- 8.13 print-uname (version number and hardware information), 188
- 9.1 bit-pos-loop.c (bit position with loop), 194
- 9.2 bit-pos-asm.c (bit position with bsrl), 195
- 10.1 simpleid.c (printing user and group IDs), 200
- 10.2 stat-perm.c (viewing file permissions with stat system call), 202
- 10.3 setuid-test.c (setuid programs), 207
- 10.4 pam.c (PAM example), 209
- 10.5 temp-file.c (temporary file creation), 214
- 10.6 grep-dictionary.c (word search), 216
- 11.1 server.h (function and variable declarations), 222
- 11.2 common.c (utility functions), 223
- 11.3 module.c (loading server modules), 226
- 11.4 server.c (server implementation), 228
- 11.5 main.c (main server program), 235
- 11.6 time.c (show wall-clock time), 239
- 11.7 issue.c (GNU/Linux distribution information), 240
- 11.8 diskfree.c (free disk space information), 242
- 11.9 processes.c (summarizing running processes), 244
- 11.10 Makefile (Makefile for sample application program), 252

- A.1 hello.c (Hello World), 260
- A.2 malloc-use.c (dynamic memory allocation), 267
- A.3 calculator.c (main calculator program), 274
- A.4 number.c (unary number implementation), 276
- A.5 stack.c (unary number stack), 279
- A.6 definitions.h (header file for calculator program), 280
- B.1 create-file.c (create a new file), 284
- B.2 timestamp.c (append a timestamp), 285
- B.3 write-all.c (write all buffered data), 286
- B.4 hexdump.c (print a hexadecimal file dump), 287
- B.5 lseek-huge.c (creating large files), 289
- B.6 read-file.c (reading files into buffers), 292
- B.7 write-args.c (writev function), 294
- B.8 listdir.c (printing directory listings), 297

## About the Authors



**Mark Mitchell** received a bachelor of arts degree in computer science from Harvard in 1994 and a master of science degree from Stanford in 1999. His research interests centered on computational complexity and computer security. Mark has participated substantially in the development of the GNU Compiler Collection, and he has a strong interest in developing quality software.



**Jeffrey Oldham** received a bachelor of arts degree in computer science from Rice University in 1991. After working at the Center for Research on Parallel Computation, he obtained a doctor of philosophy degree from Stanford in 2000. His research interests center on algorithm engineering, concentrating on flow and other combinatorial algorithms. He works on GCC and scientific computing software.



**Alex Samuel** graduated from Harvard in 1995 with a degree in physics. He worked as a software engineer at BBN before returning to study physics at Caltech and the Stanford Linear Accelerator Center. Alex administers the Software Carpentry project and works on various other projects, such as optimizations in GCC.

Mark and Alex founded **CodeSourcery LLC** together in 1999. Jeffrey joined the company in 2000. CodeSourcery's mission is to provide development tools for GNU/Linux and other operating systems; to make the GNU tool chain a commercial-quality, standards-conforming development tool set; and to provide general consulting and engineering services. CodeSourcery's Web site is <http://www.codesourcery.com>.

## About the Technical Reviewers

These reviewers contributed their considerable hands-on expertise to the entire development process for *Advanced Linux Programming*. As the book was being written, these dedicated professionals reviewed all the material for technical content, organization, and flow. Their feedback was critical to ensuring that *Advanced Linux Programming* fits our reader's need for the highest quality technical information.

**Glenn Becker** has many degrees, all in theatre. He presently works as an online producer for SCIFI.COM, the online component of the SCI FI channel, in New York City. At home he runs Debian GNU/Linux and obsesses about such topics as system administration, security, software internationalization, and XML.



**John Dean** received a BSc(Hons) from the University of Sheffield in 1974, in pure science. As an undergraduate at Sheffield, John developed his interest in computing. In 1986 he received a MSc from Cranfield Institute of Science and Technology in Control Engineering. While working for Roll Royce and Associates, John became involved in developing control software for computer-aided inspection equipment of nuclear steam-raising plants. Since leaving RR&A in 1978, he has worked in the petrochemical industry developing and maintaining process control software. John worked a volunteer software developer for MySQL from 1996 until May 2000, when he joined MySQL as a full-time employee. John's area of responsibility is MySQL on MS Windows and developing a new MySQL GUI client using Trolltech's Qt GUI application toolkit on both Windows and platforms that run X-11.



## Acknowledgments

We greatly appreciate the pioneering work of Richard Stallman, without whom there would never have been the GNU Project, and of Linus Torvalds, without whom there would never have been the Linux kernel. Countless others have worked on parts of the GNU/Linux operating system, and we thank them all.

We thank the faculties of Harvard and Rice for our undergraduate educations, and Caltech and Stanford for our graduate training. Without all who taught us, we would never have dared to teach others!

W. Richard Stevens wrote three excellent books on UNIX programming, and we have consulted them extensively. Roland McGrath, Ulrich Drepper, and many others wrote the GNU C library and its outstanding documentation.

Robert Brazile and Sam Kendall reviewed early outlines of this book and made wonderful suggestions about tone and content. Our technical editors and reviewers (especially Glenn Becker and John Dean) pointed out errors, made suggestions, and provided continuous encouragement. Of course, any errors that remain are no fault of theirs!

Thanks to Ann Quinn, of New Riders, for handling all the details involved in publishing a book; Laura Loveall, also of New Riders, for not letting us fall too far behind on our deadlines; and Stephanie Wall, also of New Riders, for encouraging us to write this book in the first place!

## Tell Us What You Think

As the reader of this book, you are the most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As the Executive Editor for the Web Development team at New Riders Publishing, I welcome your comments. You can fax, email, or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

*Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.*

When you write, please be sure to include this book's title and author, as well as your name and phone or fax number. I will carefully review your comments and share them with the author and editors who worked on the book.

Fax: 317-581-4663  
Email: [Stephanie.Wall@newriders.com](mailto:Stephanie.Wall@newriders.com)  
Mail: Stephanie Wall  
Executive Editor  
New Riders Publishing  
201 West 103<sup>rd</sup> Street  
Indianapolis, IN 46290 USA

# Introduction

GNU/Linux has taken the world of computers by storm. At one time, personal computer users were forced to choose among proprietary operating environments and applications. Users had no way of fixing or improving these programs, could not look “under the hood,” and were often forced to accept restrictive licenses. GNU/Linux and other open source systems have changed that—now PC users, administrators, and developers can choose a free operating environment complete with tools, applications, and full source code.

A great deal of the success of GNU/Linux is owed to its open source nature. Because the source code for programs is publicly available, everyone can take part in development, whether by fixing a small bug or by developing and distributing a complete major application. This opportunity has enticed thousands of capable developers worldwide to contribute new components and improvements to GNU/Linux, to the point that modern GNU/Linux systems rival the features of any proprietary system, and distributions include thousands of programs and applications spanning many CD-ROMs or DVDs.

The success of GNU/Linux has also validated much of the UNIX philosophy. Many of the application programming interfaces (APIs) introduced in AT&T and BSD UNIX variants survive in Linux and form the foundation on which programs are built. The UNIX philosophy of many small command line-oriented programs working together is the organizational principle that makes GNU/Linux so powerful. Even when these programs are wrapped in easy-to-use graphical user interfaces, the underlying commands are still available for power users and automated scripts.

A powerful GNU/Linux application harnesses the power of these APIs and commands in its inner workings. GNU/Linux’s APIs provide access to sophisticated features such as interprocess communication, multithreading, and high-performance networking. And many problems can be solved simply by assembling existing commands and programs using simple scripts.

## GNU and Linux

Where did the name GNU/Linux come from? You’ve certainly heard of Linux before, and you may have heard of the GNU Project. You may not have heard the name GNU/Linux, although you’re probably familiar with the system it refers to.

Linux is named after Linus Torvalds, the creator and original author of the *kernel* that runs a GNU/Linux system. The kernel is the program that performs the most basic functions of an operating system: It controls and interfaces with the computer’s hardware, handles allocation of memory and other resources, allows multiple programs to run at the same time, manages the file system, and so on.

The kernel by itself doesn't provide features that are useful to users. It can't even provide a simple prompt for users to enter basic commands. It provides no way for users to manage or edit files, communicate with other computers, or write other programs. These tasks require the use of a wide array of other programs, including command shells, file utilities, editors, and compilers. Many of these programs, in turn, use libraries of general-purpose functions, such as the library containing standard C library functions, which are not included in the kernel.

On GNU/Linux systems, many of these other programs and libraries are software developed as part of the GNU Project.<sup>1</sup> A great deal of this software predates the Linux kernel. The aim of the GNU Project is "to develop a complete UNIX-like operating system which is free software" (from the GNU Project Web site, <http://www.gnu.org>).

The Linux kernel and software from the GNU Project has proven to be a powerful combination. Although the combination is often called "Linux" for short, the complete system couldn't work without GNU software, any more than it could operate without the kernel. For this reason, throughout this book we'll refer to the complete system as GNU/Linux, except when we are specifically talking about the Linux kernel.

## The GNU General Public License

The source code contained in this book is covered by the GNU *General Public License* (GPL), which is listed in Appendix F, "GNU General Public License." A great deal of free software, especially GNU/Linux software, is licensed under it. For instance, the Linux kernel itself is licensed under the GPL, as are many other GNU programs and libraries you'll find in GNU/Linux distributions. If you use the source code in this book, be sure to read and understand the terms of the GPL.

The GNU Project Web site includes an extensive discussion of the GPL (<http://www.gnu.org/copyleft/>) and other free software licenses. You can find information about open source software licenses at <http://www.opensource.org/licenses/index.html>.

## Who Should Read This Book?

This book is intended for three types of readers:

- You might be a developer already experienced with programming for the GNU/Linux system, and you want to learn about some of its advanced features and capabilities. You might be interested in writing more sophisticated programs with features such as multiprocessing, multithreading, interprocess communication, and interaction with hardware devices. You might want to improve your programs by making them run faster, more reliably, and more securely, or by designing them to interact better with the rest of the GNU/Linux system.

1. GNU is a recursive acronym: It stands for "GNU's Not UNIX."

- You might be a developer experienced with another UNIX-like system who's interested in developing GNU/Linux software, too. You might already be familiar with standard APIs such as those in the POSIX specification. To develop GNU/Linux software, you need to know the peculiarities of the system, its limitations, additional capabilities, and conventions.
- You might be a developer making the transition from a non-UNIX environment, such as Microsoft's Win32 platform. You might already be familiar with the general principles of writing good software, but you need to know the specific techniques that GNU/Linux programs use to interact with the system and with each other. And you want to make sure your programs fit naturally into the GNU/Linux system and behave as users expect them to.

This book is not intended to be a comprehensive guide or reference to all aspects of GNU/Linux programming. Instead, we'll take a tutorial approach, introducing the most important concepts and techniques, and giving examples of how to use them. Section 1.5, "Finding More Information," in Chapter 1, "Getting Started," contains references to additional documentation, where you can obtain complete details about these and other aspects of GNU/Linux programming.

Because this is a book about advanced topics, we'll assume that you are already familiar with the C programming language and that you know how to use the standard C library functions in your programs. The C language is the most widely used language for developing GNU/Linux software; most of the commands and libraries that we discuss in this book, and most of the Linux kernel itself, are written in C.

The information in this book is equally applicable to C++ programs because that language is roughly a superset of C. Even if you program in another language, you'll find this information useful because C language APIs and conventions are the *lingua franca* of GNU/Linux.

If you've programmed on another UNIX-like system platform before, chances are good that you already know your way around Linux's low-level I/O functions (`open`, `read`, `stat`, and so on). These are different from the standard C library's I/O functions (`fopen`, `fprintf`, `fscanf`, and so on). Both are useful in GNU/Linux programming, and we use both sets of I/O functions throughout this book. If you're not familiar with the low-level I/O functions, jump to the end of the book and read Appendix B, "Low-Level I/O," before you start Chapter 2, "Writing Good GNU/Linux Software."

This book does not provide a general introduction to GNU/Linux systems. We assume that you already have a basic knowledge of how to interact with a GNU/Linux system and perform basic operations in graphical and command-line environments. If you're new to GNU/Linux, start with one of the many excellent introductory books, such as Michael Tolber's *Inside Linux* (New Riders Publishing, 2001).

## Conventions

This book follows a few typographical conventions:

- A new term is set in *italics* the first time it is introduced.
- Program text, functions, variables, and other “computer language” are set in a fixed-pitch font—for example, `printf ("Hello, world!\bksl n")`.
- Names of commands, files, and directories are also set in a fixed-pitch font—for example, `cd /`.
- When we show interactions with a command shell, we use `%` as the shell prompt (your shell is probably configured to use a different prompt). Everything after the prompt is what you type, while other lines of text are the system's response.

For example, in this interaction

```
% uname
Linux
```

the system prompted you with `%`. You entered the `uname` command. The system responded by printing `Linux`.

- The title of each source code listing includes a filename in parentheses. If you type in the listing, save it to a file by this name. You can also download the source code listings from the *Advanced Linux Programming* Web site (<http://www.newriders.com> or <http://www.advancedlinuxprogramming.com>).

We wrote this book and developed the programs listed in it using the Red Hat 6.2 distribution of GNU/Linux. This distribution incorporates release 2.2.14 of the Linux kernel, release 2.1.3 of the GNU C library, and the EGCS 1.1.2 release of the GNU C compiler. The information and programs in this book should generally be applicable to other versions and distributions of GNU/Linux as well, including 2.4 releases of the Linux kernel and 2.2 releases of the GNU C library.

