CONTENTS IN DETAIL

PREFACE **xxxi**

1 HISTORY AND STANDARDS **1**

1.1 A Brief History of UNIX and C 2
1.2 A Brief History of Linux 5
1.2.1 The GNU Project 5
1.2.2 The Linux Kernel 6
1.3 Standardization 10
1.3.1 The C Programming Language 10
1.3.2 The First POSIX Standards 11
1.3.3 X/Open Company and The Open Group 13
1.3.4 SUSv3 and POSIX.1-2001 13
1.3.5 SUSv4 and POSIX.1-2008 15
1.3.6 UNIX Standards Timeline 16
1.3.7 Implementation Standards 17
1.3.8 Linux, Standards, and the Linux Standard Base 18
1.4 Summary 19

2 FUNDAMENTAL CONCEPTS **21**

2.1 The Core Operating System: The Kernel 21
2.2 The Shell 24
2.3 Users and Groups 26
2.4 Single Directory Hierarchy, Directories, Links, and Files 27
2.5 File I/O Model 29
2.6 Programs 30
2.7 Processes 31
2.8 Memory Mappings 35
2.9 Static and Shared Libraries 35
2.10 Interprocess Communication and Synchronization 36
2.11 Signals 37
2.12 Threads 38
2.13 Process Groups and Shell Job Control 38
2.14 Sessions, Controlling Terminals, and Controlling Processes 39
2.15 Pseudoterminals 39
2.16 Date and Time 40
2.17 Client-Server Architecture 40
2.18 Realtime 41
2.19 The `/proc` File System 42
2.20 Summary 42

3 SYSTEM PROGRAMMING CONCEPTS **43**

3.1 System Calls 43
3.2 Library Functions 46
3.3 The Standard C Library; The GNU C Library (*glibc*) 47
3.4 Handling Errors from System Calls and Library Functions 48
3.5 Notes on the Example Programs in This Book 50
3.5.1 Command-Line Options and Arguments 50
3.5.2 Common Functions and Header Files 51

3.6 Portability Issues ... 61
3.6.1 Feature Test Macros ... 61
3.6.2 System Data Types ... 63
3.6.3 Miscellaneous Portability Issues ... 66
3.7 Summary ... 68
3.8 Exercise ... 68

4 FILE I/O: THE UNIVERSAL I/O MODEL 69

4.1 Overview ... 69
4.2 Universality of I/O ... 72
4.3 Opening a File: *open()* ... 72
4.3.1 The *open() flags* Argument ... 74
4.3.2 Errors from *open()* ... 77
4.3.3 The *creat()* System Call ... 78
4.4 Reading from a File: *read()* ... 79
4.5 Writing to a File: *write()* ... 80
4.6 Closing a File: *close()* ... 80
4.7 Changing the File Offset: *lseek()* ... 81
4.8 Operations Outside the Universal I/O Model: *ioctl()* ... 86
4.9 Summary ... 86
4.10 Exercises ... 87

5 FILE I/O: FURTHER DETAILS 89

5.1 Atomicity and Race Conditions ... 90
5.2 File Control Operations: *fcntl()* ... 92
5.3 Open File Status Flags ... 93
5.4 Relationship Between File Descriptors and Open Files ... 94
5.5 Duplicating File Descriptors ... 96
5.6 File I/O at a Specified Offset: *pread()* and *pwrite()* ... 98
5.7 Scatter-Gather I/O: *readv()* and *writev()* ... 99
5.8 Truncating a File: *truncate()* and *ftruncate()* ... 103
5.9 Nonblocking I/O ... 103
5.10 I/O on Large Files ... 104
5.11 The `/dev/fd` Directory ... 107
5.12 Creating Temporary Files ... 108
5.13 Summary ... 109
5.14 Exercises ... 110

6 PROCESSES 113

6.1 Processes and Programs ... 113
6.2 Process ID and Parent Process ID ... 114
6.3 Memory Layout of a Process ... 115
6.4 Virtual Memory Management ... 118
6.5 The Stack and Stack Frames ... 121
6.6 Command-Line Arguments (*argc*, *argv*) ... 122
6.7 Environment List ... 125
6.8 Performing a Nonlocal Goto: *setjmp()* and *longjmp()* ... 131
6.9 Summary ... 138
6.10 Exercises ... 138

TLPI_FINAL_RRD.pdf 12 8/5/2010 10:46:56 AM

7 MEMORY ALLOCATION 139

7.1 Allocating Memory on the Heap ... 139
7.1.1 Adjusting the Program Break: *brk()* and *sbrk()* ... 139
7.1.2 Allocating Memory on the Heap: *malloc()* and *free()* ... 140
7.1.3 Implementation of *malloc()* and *free()* ... 144
7.1.4 Other Methods of Allocating Memory on the Heap ... 147
7.2 Allocating Memory on the Stack: *alloca()* ... 150
7.3 Summary ... 151
7.4 Exercises ... 152

8 USERS AND GROUPS 153

8.1 The Password File: `/etc/passwd` ... 153
8.2 The Shadow Password File: `/etc/shadow` ... 155
8.3 The Group File: `/etc/group` ... 155
8.4 Retrieving User and Group Information ... 157
8.5 Password Encryption and User Authentication ... 162
8.6 Summary ... 166
8.7 Exercises ... 166

9 PROCESS CREDENTIALS 167

9.1 Real User ID and Real Group ID ... 167
9.2 Effective User ID and Effective Group ID ... 168
9.3 Set-User-ID and Set-Group-ID Programs ... 168
9.4 Saved Set-User-ID and Saved Set-Group-ID ... 170
9.5 File-System User ID and File-System Group ID ... 171
9.6 Supplementary Group IDs ... 172
9.7 Retrieving and Modifying Process Credentials ... 172
9.7.1 Retrieving and Modifying Real, Effective, and Saved Set IDs ... 172
9.7.2 Retrieving and Modifying File-System IDs ... 178
9.7.3 Retrieving and Modifying Supplementary Group IDs ... 178
9.7.4 Summary of Calls for Modifying Process Credentials ... 180
9.7.5 Example: Displaying Process Credentials ... 182
9.8 Summary ... 183
9.9 Exercises ... 184

10 TIME 185

10.1 Calendar Time ... 186
10.2 Time-Conversion Functions ... 187
10.2.1 Converting *time_t* to Printable Form ... 188
10.2.2 Converting Between *time_t* and Broken-Down Time ... 189
10.2.3 Converting Between Broken-Down Time and Printable Form ... 191
10.3 Timezones ... 197
10.4 Locales ... 200
10.5 Updating the System Clock ... 204
10.6 The Software Clock (Jiffies) ... 205
10.7 Process Time ... 206
10.8 Summary ... 209
10.9 Exercise ... 210

TLPI_FINAL_RRD.pdf 13 8/5/2010 10:46:56 AM

11 SYSTEM LIMITS AND OPTIONS 211
11.1 System Limits ... 212
11.2 Retrieving System Limits (and Options) at Run Time ... 215
11.3 Retrieving File-Related Limits (and Options) at Run Time ... 217
11.4 Indeterminate Limits ... 219
11.5 System Options ... 219
11.6 Summary ... 221
11.7 Exercises ... 222

12 SYSTEM AND PROCESS INFORMATION 223
12.1 The /proc File System ... 223
12.1.1 Obtaining Information About a Process: /proc/*PID* ... 224
12.1.2 System Information Under /proc ... 226
12.1.3 Accessing /proc Files ... 226
12.2 System Identification: *uname()* ... 229
12.3 Summary ... 231
12.4 Exercises ... 231

13 FILE I/O BUFFERING 233
13.1 Kernel Buffering of File I/O: The Buffer Cache ... 233
13.2 Buffering in the *stdio* Library ... 237
13.3 Controlling Kernel Buffering of File I/O ... 239
13.4 Summary of I/O Buffering ... 243
13.5 Advising the Kernel About I/O Patterns ... 244
13.6 Bypassing the Buffer Cache: Direct I/O ... 246
13.7 Mixing Library Functions and System Calls for File I/O ... 248
13.8 Summary ... 249
13.9 Exercises ... 250

14 FILE SYSTEMS 251
14.1 Device Special Files (Devices) ... 252
14.2 Disks and Partitions ... 253
14.3 File Systems ... 254
14.4 I-nodes ... 256
14.5 The Virtual File System (VFS) ... 259
14.6 Journaling File Systems ... 260
14.7 Single Directory Hierarchy and Mount Points ... 261
14.8 Mounting and Unmounting File Systems ... 262
14.8.1 Mounting a File System: *mount()* ... 264
14.8.2 Unmounting a File System: *umount()* and *umount2()* ... 269
14.9 Advanced Mount Features ... 271
14.9.1 Mounting a File System at Multiple Mount Points ... 271
14.9.2 Stacking Multiple Mounts on the Same Mount Point ... 271
14.9.3 Mount Flags That Are Per-Mount Options ... 272
14.9.4 Bind Mounts ... 272
14.9.5 Recursive Bind Mounts ... 273
14.10 A Virtual Memory File System: *tmpfs* ... 274
14.11 Obtaining Information About a File System: *statvfs()* ... 276
14.12 Summary ... 277
14.13 Exercise ... 278

TLPI_FINAL_RRD.pdf 14 8/5/2010 10:46:56 AM

15 FILE ATTRIBUTES 279
15.1 Retrieving File Information: *stat()* ... 279
15.2 File Timestamps ... 285
15.2.1 Changing File Timestamps with *utime()* and *utimes()* ... 287
15.2.2 Changing File Timestamps with *utimensat()* and *futimens()* ... 289
15.3 File Ownership ... 291
15.3.1 Ownership of New Files ... 291
15.3.2 Changing File Ownership: *chown()*, *fchown()*, and *lchown()* ... 291
15.4 File Permissions ... 294
15.4.1 Permissions on Regular Files ... 294
15.4.2 Permissions on Directories ... 297
15.4.3 Permission-Checking Algorithm ... 297
15.4.4 Checking File Accessibility: *access()* ... 299
15.4.5 Set-User-ID, Set-Group-ID, and Sticky Bits ... 300
15.4.6 The Process File Mode Creation Mask: *umask()* ... 301
15.4.7 Changing File Permissions: *chmod()* and *fchmod()* ... 303
15.5 I-node Flags (*ext2* Extended File Attributes) ... 304
15.6 Summary ... 308
15.7 Exercises ... 309

16 EXTENDED ATTRIBUTES 311
16.1 Overview ... 311
16.2 Extended Attribute Implementation Details ... 313
16.3 System Calls for Manipulating Extended Attributes ... 314
16.4 Summary ... 318
16.5 Exercise ... 318

17 ACCESS CONTROL LISTS 319
17.1 Overview ... 320
17.2 ACL Permission-Checking Algorithm ... 321
17.3 Long and Short Text Forms for ACLs ... 323
17.4 The `ACL_MASK` Entry and the ACL Group Class ... 324
17.5 The *getfacl* and *setfacl* Commands ... 325
17.6 Default ACLs and File Creation ... 327
17.7 ACL Implementation Limits ... 328
17.8 The ACL API ... 329
17.9 Summary ... 337
17.10 Exercise ... 337

18 DIRECTORIES AND LINKS 339
18.1 Directories and (Hard) Links ... 339
18.2 Symbolic (Soft) Links ... 342
18.3 Creating and Removing (Hard) Links: *link()* and *unlink()* ... 344
18.4 Changing the Name of a File: *rename()* ... 348
18.5 Working with Symbolic Links: *symlink()* and *readlink()* ... 349
18.6 Creating and Removing Directories: *mkdir()* and *rmdir()* ... 350
18.7 Removing a File or Directory: *remove()* ... 352
18.8 Reading Directories: *opendir()* and *readdir()* ... 352
18.9 File Tree Walking: *nftw()* ... 358
18.10 The Current Working Directory of a Process ... 363
18.11 Operating Relative to a Directory File Descriptor ... 365
18.12 Changing the Root Directory of a Process: *chroot()* ... 367
18.13 Resolving a Pathname: *realpath()* ... 369

TLPI_FINAL_RRD.pdf 15 8/5/2010 10:46:56 AM

18.14 Parsing Pathname Strings: *dirname()* and *basename()* 370
18.15 Summary 372
18.16 Exercises 373

19 MONITORING FILE EVENTS 375

19.1 Overview 376
19.2 The *inotify* API 376
19.3 *inotify* Events 378
19.4 Reading *inotify* Events 379
19.5 Queue Limits and `/proc` Files 385
19.6 An Older System for Monitoring File Events: *dnotify* 386
19.7 Summary 386
19.8 Exercise 386

20 SIGNALS: FUNDAMENTAL CONCEPTS 387

20.1 Concepts and Overview 388
20.2 Signal Types and Default Actions 390
20.3 Changing Signal Dispositions: *signal()* 397
20.4 Introduction to Signal Handlers 398
20.5 Sending Signals: *kill()* 401
20.6 Checking for the Existence of a Process 403
20.7 Other Ways of Sending Signals: *raise()* and *killpg()* 404
20.8 Displaying Signal Descriptions 406
20.9 Signal Sets 406
20.10 The Signal Mask (Blocking Signal Delivery) 410
20.11 Pending Signals 411
20.12 Signals Are Not Queued 412
20.13 Changing Signal Dispositions: *sigaction()* 416
20.14 Waiting for a Signal: *pause()* 418
20.15 Summary 418
20.16 Exercises 419

21 SIGNALS: SIGNAL HANDLERS 421

21.1 Designing Signal Handlers 422
21.1.1 Signals Are Not Queued (Revisited) 422
21.1.2 Reentrant and Async-Signal-Safe Functions 422
21.1.3 Global Variables and the *sig_atomic_t* Data Type 428
21.2 Other Methods of Terminating a Signal Handler 428
21.2.1 Performing a Nonlocal Goto from a Signal Handler 429
21.2.2 Terminating a Process Abnormally: *abort()* 433
21.3 Handling a Signal on an Alternate Stack: *sigaltstack()* 434
21.4 The `SA_SIGINFO` Flag 437
21.5 Interruption and Restarting of System Calls 442
21.6 Summary 445
21.7 Exercise 446

22 SIGNALS: ADVANCED FEATURES 447

22.1 Core Dump Files 448
22.2 Special Cases for Delivery, Disposition, and Handling 450
22.3 Interruptible and Uninterruptible Process Sleep States 451
22.4 Hardware-Generated Signals 452
22.5 Synchronous and Asynchronous Signal Generation 452

TLPI_FINAL_RRD.pdf 16 8/5/2010 10:46:56 AM

22.6 Timing and Order of Signal Delivery ... 453
22.7 Implementation and Portability of *signal()* ... 454
22.8 Realtime Signals ... 456
22.8.1 Sending Realtime Signals ... 458
22.8.2 Handling Realtime Signals ... 460
22.9 Waiting for a Signal Using a Mask: *sigsuspend()* ... 464
22.10 Synchronously Waiting for a Signal ... 468
22.11 Fetching Signals via a File Descriptor ... 471
22.12 Interprocess Communication with Signals ... 474
22.13 Earlier Signal APIs (System V and BSD) ... 475
22.14 Summary ... 477
22.15 Exercises ... 478

23 TIMERS AND SLEEPING 479

23.1 Interval Timers ... 479
23.2 Scheduling and Accuracy of Timers ... 485
23.3 Setting Timeouts on Blocking Operations ... 486
23.4 Suspending Execution for a Fixed Interval (Sleeping) ... 487
23.4.1 Low-Resolution Sleeping: *sleep()* ... 487
23.4.2 High-Resolution Sleeping: *nanosleep()* ... 488
23.5 POSIX Clocks ... 491
23.5.1 Retrieving the Value of a Clock: *clock_gettime()* ... 491
23.5.2 Setting the Value of a Clock: *clock_settime()* ... 492
23.5.3 Obtaining the Clock ID of a Specific Process or Thread ... 493
23.5.4 Improved High-Resolution Sleeping: *clock_nanosleep()* ... 493
23.6 POSIX Interval Timers ... 495
23.6.1 Creating a Timer: *timer_create()* ... 495
23.6.2 Arming and Disarming a Timer: *timer_settime()* ... 498
23.6.3 Retrieving the Current Value of a Timer: *timer_gettime()* ... 499
23.6.4 Deleting a Timer: *timer_delete()* ... 499
23.6.5 Notification via a Signal ... 499
23.6.6 Timer Overruns ... 503
23.6.7 Notification via a Thread ... 504
23.7 Timers That Notify via File Descriptors: the *timerfd* API ... 507
23.8 Summary ... 511
23.9 Exercises ... 512

24 PROCESS CREATION 513

24.1 Overview of *fork()*, *exit()*, *wait()*, and *execve()* ... 513
24.2 Creating a New Process: *fork()* ... 515
24.2.1 File Sharing Between Parent and Child ... 517
24.2.2 Memory Semantics of *fork()* ... 520
24.3 The *vfork()* System Call ... 522
24.4 Race Conditions After *fork()* ... 525
24.5 Avoiding Race Conditions by Synchronizing with Signals ... 527
24.6 Summary ... 529
24.7 Exercises ... 530

25 PROCESS TERMINATION 531

25.1 Terminating a Process: *_exit()* and *exit()* ... 531
25.2 Details of Process Termination ... 533
25.3 Exit Handlers ... 533
25.4 Interactions Between *fork()*, *stdio* Buffers, and *_exit()* ... 537

TLPI_FINAL_RRD.pdf 17 8/5/2010 10:46:56 AM

25.5 Summary ... 538
25.6 Exercise ... 539

26 MONITORING CHILD PROCESSES 541

26.1 Waiting on a Child Process ... 541
26.1.1 The *wait()* System Call ... 541
26.1.2 The *waitpid()* System Call ... 544
26.1.3 The Wait Status Value ... 545
26.1.4 Process Termination from a Signal Handler ... 549
26.1.5 The *waitid()* System Call ... 550
26.1.6 The *wait3()* and *wait4()* System Calls ... 552
26.2 Orphans and Zombies ... 553
26.3 The SIGCHLD Signal ... 555
26.3.1 Establishing a Handler for SIGCHLD ... 555
26.3.2 Delivery of SIGCHLD for Stopped Children ... 559
26.3.3 Ignoring Dead Child Processes ... 559
26.4 Summary ... 561
26.5 Exercises ... 562

27 PROGRAM EXECUTION 563

27.1 Executing a New Program: *execve()* ... 563
27.2 The *exec()* Library Functions ... 567
27.2.1 The PATH Environment Variable ... 568
27.2.2 Specifying Program Arguments as a List ... 570
27.2.3 Passing the Caller's Environment to the New Program ... 570
27.2.4 Executing a File Referred to by a Descriptor: *fexecve()* ... 571
27.3 Interpreter Scripts ... 572
27.4 File Descriptors and *exec()* ... 575
27.5 Signals and *exec()* ... 578
27.6 Executing a Shell Command: *system()* ... 579
27.7 Implementing *system()* ... 582
27.8 Summary ... 588
27.9 Exercises ... 589

28 PROCESS CREATION AND PROGRAM EXECUTION IN MORE DETAIL 591

28.1 Process Accounting ... 591
28.2 The *clone()* System Call ... 598
28.2.1 The *clone() flags* Argument ... 603
28.2.2 Extensions to *waitpid()* for Cloned Children ... 609
28.3 Speed of Process Creation ... 610
28.4 Effect of *exec()* and *fork()* on Process Attributes ... 612
28.5 Summary ... 616
28.6 Exercise ... 616

29 THREADS: INTRODUCTION 617

29.1 Overview ... 617
29.2 Background Details of the Pthreads API ... 620
29.3 Thread Creation ... 622
29.4 Thread Termination ... 623
29.5 Thread IDs ... 624
29.6 Joining with a Terminated Thread ... 625
29.7 Detaching a Thread ... 627

TLPI_FINAL_RRD.pdf 18 8/5/2010 10:46:56 AM

29.8 Thread Attributes ... 628
29.9 Threads Versus Processes ... 629
29.10 Summary ... 629
29.11 Exercises ... 630

30 THREADS: THREAD SYNCHRONIZATION 631

30.1 Protecting Accesses to Shared Variables: Mutexes ... 631
30.1.1 Statically Allocated Mutexes ... 635
30.1.2 Locking and Unlocking a Mutex ... 635
30.1.3 Performance of Mutexes ... 638
30.1.4 Mutex Deadlocks ... 639
30.1.5 Dynamically Initializing a Mutex ... 639
30.1.6 Mutex Attributes ... 640
30.1.7 Mutex Types ... 640
30.2 Signaling Changes of State: Condition Variables ... 642
30.2.1 Statically Allocated Condition Variables ... 643
30.2.2 Signaling and Waiting on Condition Variables ... 643
30.2.3 Testing a Condition Variable's Predicate ... 647
30.2.4 Example Program: Joining Any Terminated Thread ... 648
30.2.5 Dynamically Allocated Condition Variables ... 651
30.3 Summary ... 652
30.4 Exercises ... 652

31 THREADS: THREAD SAFETY AND PER-THREAD STORAGE 655

31.1 Thread Safety (and Reentrancy Revisited) ... 655
31.2 One-Time Initialization ... 658
31.3 Thread-Specific Data ... 659
31.3.1 Thread-Specific Data from the Library Function's Perspective ... 660
31.3.2 Overview of the Thread-Specific Data API ... 660
31.3.3 Details of the Thread-Specific Data API ... 661
31.3.4 Employing the Thread-Specific Data API ... 663
31.3.5 Thread-Specific Data Implementation Limits ... 668
31.4 Thread-Local Storage ... 668
31.5 Summary ... 669
31.6 Exercises ... 670

32 THREADS: THREAD CANCELLATION 671

32.1 Canceling a Thread ... 671
32.2 Cancellation State and Type ... 672
32.3 Cancellation Points ... 673
32.4 Testing for Thread Cancellation ... 675
32.5 Cleanup Handlers ... 676
32.6 Asynchronous Cancelability ... 680
32.7 Summary ... 680

33 THREADS: FURTHER DETAILS 681

33.1 Thread Stacks ... 681
33.2 Threads and Signals ... 682
33.2.1 How the UNIX Signal Model Maps to Threads ... 682
33.2.2 Manipulating the Thread Signal Mask ... 684
33.2.3 Sending a Signal to a Thread ... 684
33.2.4 Dealing with Asynchronous Signals Sanely ... 685

TLPI_FINAL_RRD.pdf 19 8/5/2010 10:46:56 AM

33.3 Threads and Process Control ... 686
33.4 Thread Implementation Models ... 687
33.5 Linux Implementations of POSIX Threads ... 689
33.5.1 LinuxThreads ... 689
33.5.2 NPTL ... 692
33.5.3 Which Threading Implementation? ... 694
33.6 Advanced Features of the Pthreads API ... 696
33.7 Summary ... 696
33.8 Exercises ... 697

34 PROCESS GROUPS, SESSIONS, AND JOB CONTROL 699

34.1 Overview ... 699
34.2 Process Groups ... 701
34.3 Sessions ... 704
34.4 Controlling Terminals and Controlling Processes ... 706
34.5 Foreground and Background Process Groups ... 708
34.6 The SIGHUP Signal ... 709
34.6.1 Handling of SIGHUP by the Shell ... 710
34.6.2 SIGHUP and Termination of the Controlling Process ... 712
34.7 Job Control ... 714
34.7.1 Using Job Control Within the Shell ... 714
34.7.2 Implementing Job Control ... 717
34.7.3 Handling Job-Control Signals ... 722
34.7.4 Orphaned Process Groups (and SIGHUP Revisited) ... 725
34.8 Summary ... 730
34.9 Exercises ... 731

35 PROCESS PRIORITIES AND SCHEDULING 733

35.1 Process Priorities (Nice Values) ... 733
35.2 Overview of Realtime Process Scheduling ... 737
35.2.1 The SCHED_RR Policy ... 739
35.2.2 The SCHED_FIFO Policy ... 740
35.2.3 The SCHED_BATCH and SCHED_IDLE Policies ... 740
35.3 Realtime Process Scheduling API ... 740
35.3.1 Realtime Priority Ranges ... 740
35.3.2 Modifying and Retrieving Policies and Priorities ... 741
35.3.3 Relinquishing the CPU ... 747
35.3.4 The SCHED_RR Time Slice ... 747
35.4 CPU Affinity ... 748
35.5 Summary ... 751
35.6 Exercises ... 751

36 PROCESS RESOURCES 753

36.1 Process Resource Usage ... 753
36.2 Process Resource Limits ... 755
36.3 Details of Specific Resource Limits ... 760
36.4 Summary ... 765
36.5 Exercises ... 765

37 DAEMONS 767

37.1 Overview ... 767
37.2 Creating a Daemon ... 768

TLPI_FINAL_RRD.pdf 20 8/5/2010 10:46:56 AM

37.3 Guidelines for Writing Daemons ... 771
37.4 Using SIGHUP to Reinitialize a Daemon ... 772
37.5 Logging Messages and Errors Using *syslog* ... 775
37.5.1 Overview ... 775
37.5.2 The *syslog* API ... 777
37.5.3 The /etc/syslog.conf File ... 781
37.6 Summary ... 782
37.7 Exercise ... 782

38 WRITING SECURE PRIVILEGED PROGRAMS 783

38.1 Is a Set-User-ID or Set-Group-ID Program Required? ... 784
38.2 Operate with Least Privilege ... 784
38.3 Be Careful When Executing a Program ... 787
38.4 Avoid Exposing Sensitive Information ... 788
38.5 Confine the Process ... 789
38.6 Beware of Signals and Race Conditions ... 790
38.7 Pitfalls When Performing File Operations and File I/O ... 790
38.8 Don't Trust Inputs or the Environment ... 791
38.9 Beware of Buffer Overruns ... 792
38.10 Beware of Denial-of-Service Attacks ... 793
38.11 Check Return Statuses and Fail Safely ... 794
38.12 Summary ... 795
38.13 Exercises ... 796

39 CAPABILITIES 797

39.1 Rationale for Capabilities ... 797
39.2 The Linux Capabilities ... 798
39.3 Process and File Capabilities ... 798
39.3.1 Process Capabilities ... 798
39.3.2 File Capabilities ... 799
39.3.3 Purpose of the Process Permitted and Effective Capability Sets ... 802
39.3.4 Purpose of the File Permitted and Effective Capability Sets ... 802
39.3.5 Purpose of the Process and File Inheritable Sets ... 802
39.3.6 Assigning and Viewing File Capabilities from the Shell ... 803
39.4 The Modern Capabilities Implementation ... 804
39.5 Transformation of Process Capabilities During *exec()* ... 805
39.5.1 Capability Bounding Set ... 805
39.5.2 Preserving *root* Semantics ... 806
39.6 Effect on Process Capabilities of Changing User IDs ... 806
39.7 Changing Process Capabilities Programmatically ... 807
39.8 Creating Capabilities-Only Environments ... 811
39.9 Discovering the Capabilities Required by a Program ... 813
39.10 Older Kernels and Systems Without File Capabilities ... 814
39.11 Summary ... 816
39.12 Exercise ... 816

40 LOGIN ACCOUNTING 817

40.1 Overview of the utmp and wtmp Files ... 817
40.2 The *utmpx* API ... 818
40.3 The *utmpx* Structure ... 818
40.4 Retrieving Information from the utmp and wtmp Files ... 821
40.5 Retrieving the Login Name: *getlogin()* ... 825
40.6 Updating the utmp and wtmp Files for a Login Session ... 825

TLPI_FINAL_RRD.pdf 21 8/5/2010 10:46:56 AM

40.7 The lastlog File 830
40.8 Summary 832
40.9 Exercises 832

41 FUNDAMENTALS OF SHARED LIBRARIES 833

41.1 Object Libraries 833
41.2 Static Libraries 834
41.3 Overview of Shared Libraries 836
41.4 Creating and Using Shared Libraries—A First Pass 837
41.4.1 Creating a Shared Library 837
41.4.2 Position-Independent Code 838
41.4.3 Using a Shared Library 839
41.4.4 The Shared Library Soname 840
41.5 Useful Tools for Working with Shared Libraries 843
41.6 Shared Library Versions and Naming Conventions 844
41.7 Installing Shared Libraries 847
41.8 Compatible Versus Incompatible Libraries 850
41.9 Upgrading Shared Libraries 850
41.10 Specifying Library Search Directories in an Object File 851
41.11 Finding Shared Libraries at Run Time 854
41.12 Run-Time Symbol Resolution 854
41.13 Using a Static Library Instead of a Shared Library 856
41.14 Summary 856
41.15 Exercise 857

42 ADVANCED FEATURES OF SHARED LIBRARIES 859

42.1 Dynamically Loaded Libraries 859
42.1.1 Opening a Shared Library: *dlopen()* 860
42.1.2 Diagnosing Errors: *dlerror()* 862
42.1.3 Obtaining the Address of a Symbol: *dlsym()* 862
42.1.4 Closing a Shared Library: *dlclose()* 866
42.1.5 Obtaining Information About Loaded Symbols: *dladdr()* 866
42.1.6 Accessing Symbols in the Main Program 867
42.2 Controlling Symbol Visibility 867
42.3 Linker Version Scripts 868
42.3.1 Controlling Symbol Visibility with Version Scripts 868
42.3.2 Symbol Versioning 870
42.4 Initialization and Finalization Functions 872
42.5 Preloading Shared Libraries 873
42.6 Monitoring the Dynamic Linker: LD_DEBUG 874
42.7 Summary 875
42.8 Exercises 876

43 INTERPROCESS COMMUNICATION OVERVIEW 877

43.1 A Taxonomy of IPC Facilities 877
43.2 Communication Facilities 879
43.3 Synchronization Facilities 880
43.4 Comparing IPC Facilities 882
43.5 Summary 887
43.6 Exercises 887

TLPI_FINAL_RRD.pdf 22 8/5/2010 10:46:56 AM

44 PIPES AND FIFOS 889
44.1 Overview ... 889
44.2 Creating and Using Pipes ... 892
44.3 Pipes as a Method of Process Synchronization ... 897
44.4 Using Pipes to Connect Filters ... 899
44.5 Talking to a Shell Command via a Pipe: *popen()* ... 902
44.6 Pipes and *stdio* Buffering ... 906
44.7 FIFOs ... 906
44.8 A Client-Server Application Using FIFOs ... 909
44.9 Nonblocking I/O ... 915
44.10 Semantics of *read()* and *write()* on Pipes and FIFOs ... 917
44.11 Summary ... 918
44.12 Exercises ... 919

45 INTRODUCTION TO SYSTEM V IPC 921
45.1 API Overview ... 922
45.2 IPC Keys ... 925
45.3 Associated Data Structure and Object Permissions ... 927
45.4 IPC Identifiers and Client-Server Applications ... 929
45.5 Algorithm Employed by System V IPC *get* Calls ... 931
45.6 The *ipcs* and *ipcrm* Commands ... 934
45.7 Obtaining a List of All IPC Objects ... 935
45.8 IPC Limits ... 935
45.9 Summary ... 936
45.10 Exercises ... 936

46 SYSTEM V MESSAGE QUEUES 937
46.1 Creating or Opening a Message Queue ... 938
46.2 Exchanging Messages ... 940
46.2.1 Sending Messages ... 940
46.2.2 Receiving Messages ... 943
46.3 Message Queue Control Operations ... 947
46.4 Message Queue Associated Data Structure ... 948
46.5 Message Queue Limits ... 950
46.6 Displaying All Message Queues on the System ... 951
46.7 Client-Server Programming with Message Queues ... 953
46.8 A File-Server Application Using Message Queues ... 955
46.9 Disadvantages of System V Message Queues ... 961
46.10 Summary ... 962
46.11 Exercises ... 963

47 SYSTEM V SEMAPHORES 965
47.1 Overview ... 966
47.2 Creating or Opening a Semaphore Set ... 969
47.3 Semaphore Control Operations ... 969
47.4 Semaphore Associated Data Structure ... 972
47.5 Semaphore Initialization ... 975
47.6 Semaphore Operations ... 978
47.7 Handling of Multiple Blocked Semaphore Operations ... 986
47.8 Semaphore Undo Values ... 986
47.9 Implementing a Binary Semaphores Protocol ... 988

TLPI_FINAL_RRD.pdf 23 8/5/2010 10:46:56 AM

47.10 Semaphore Limits 991
47.11 Disadvantages of System V Semaphores 993
47.12 Summary 993
47.13 Exercises 994

48 SYSTEM V SHARED MEMORY 997

48.1 Overview 998
48.2 Creating or Opening a Shared Memory Segment 998
48.3 Using Shared Memory 999
48.4 Example: Transferring Data via Shared Memory 1001
48.5 Location of Shared Memory in Virtual Memory 1006
48.6 Storing Pointers in Shared Memory 1010
48.7 Shared Memory Control Operations 1011
48.8 Shared Memory Associated Data Structure 1012
48.9 Shared Memory Limits 1014
48.10 Summary 1015
48.11 Exercises 1016

49 MEMORY MAPPINGS 1017

49.1 Overview 1017
49.2 Creating a Mapping: *mmap()* 1020
49.3 Unmapping a Mapped Region: *munmap()* 1023
49.4 File Mappings 1024
49.4.1 Private File Mappings 1024
49.4.2 Shared File Mappings 1025
49.4.3 Boundary Cases 1029
49.4.4 Memory Protection and File Access Mode Interactions 1030
49.5 Synchronizing a Mapped Region: *msync()* 1031
49.6 Additional *mmap()* Flags 1033
49.7 Anonymous Mappings 1034
49.8 Remapping a Mapped Region: *mremap()* 1037
49.9 `MAP_NORESERVE` and Swap Space Overcommitting 1038
49.10 The `MAP_FIXED` Flag 1040
49.11 Nonlinear Mappings: *remap_file_pages()* 1041
49.12 Summary 1043
49.13 Exercises 1044

50 VIRTUAL MEMORY OPERATIONS 1045

50.1 Changing Memory Protection: *mprotect()* 1045
50.2 Memory Locking: *mlock()* and *mlockall()* 1047
50.3 Determining Memory Residence: *mincore()* 1051
50.4 Advising Future Memory Usage Patterns: *madvise()* 1054
50.5 Summary 1056
50.6 Exercises 1056

51 INTRODUCTION TO POSIX IPC 1057

51.1 API Overview 1058
51.2 Comparison of System V IPC and POSIX IPC 1061
51.3 Summary 1062

TLPI_FINAL_RRD.pdf 24 8/5/2010 10:46:56 AM

52 POSIX MESSAGE QUEUES 1063

52.1 Overview ... 1064
52.2 Opening, Closing, and Unlinking a Message Queue ... 1064
52.3 Relationship Between Descriptors and Message Queues ... 1067
52.4 Message Queue Attributes ... 1068
52.5 Exchanging Messages ... 1073
52.5.1 Sending Messages ... 1073
52.5.2 Receiving Messages ... 1074
52.5.3 Sending and Receiving Messages with a Timeout ... 1077
52.6 Message Notification ... 1077
52.6.1 Receiving Notification via a Signal ... 1079
52.6.2 Receiving Notification via a Thread ... 1082
52.7 Linux-Specific Features ... 1083
52.8 Message Queue Limits ... 1085
52.9 Comparison of POSIX and System V Message Queues ... 1086
52.10 Summary ... 1087
52.11 Exercises ... 1087

53 POSIX SEMAPHORES 1089

53.1 Overview ... 1089
53.2 Named Semaphores ... 1090
53.2.1 Opening a Named Semaphore ... 1090
53.2.2 Closing a Semaphore ... 1093
53.2.3 Removing a Named Semaphore ... 1093
53.3 Semaphore Operations ... 1094
53.3.1 Waiting on a Semaphore ... 1094
53.3.2 Posting a Semaphore ... 1096
53.3.3 Retrieving the Current Value of a Semaphore ... 1097
53.4 Unnamed Semaphores ... 1099
53.4.1 Initializing an Unnamed Semaphore ... 1100
53.4.2 Destroying an Unnamed Semaphore ... 1102
53.5 Comparisons with Other Synchronization Techniques ... 1103
53.6 Semaphore Limits ... 1104
53.7 Summary ... 1105
53.8 Exercises ... 1105

54 POSIX SHARED MEMORY 1107

54.1 Overview ... 1108
54.2 Creating Shared Memory Objects ... 1109
54.3 Using Shared Memory Objects ... 1112
54.4 Removing Shared Memory Objects ... 1114
54.5 Comparisons Between Shared Memory APIs ... 1115
54.6 Summary ... 1116
54.7 Exercise ... 1116

55 FILE LOCKING 1117

55.1 Overview ... 1117
55.2 File Locking with *flock()* ... 1119
55.2.1 Semantics of Lock Inheritance and Release ... 1122
55.2.2 Limitations of *flock()* ... 1123

TLPI_FINAL_RRD.pdf 25 8/5/2010 10:46:57 AM

55.3 Record Locking with *fcntl()* ... 1124
55.3.1 Deadlock ... 1128
55.3.2 Example: An Interactive Locking Program ... 1129
55.3.3 Example: A Library of Locking Functions ... 1133
55.3.4 Lock Limits and Performance ... 1135
55.3.5 Semantics of Lock Inheritance and Release ... 1136
55.3.6 Lock Starvation and Priority of Queued Lock Requests ... 1137
55.4 Mandatory Locking ... 1137
55.5 The `/proc/locks` File ... 1140
55.6 Running Just One Instance of a Program ... 1142
55.7 Older Locking Techniques ... 1144
55.8 Summary ... 1146
55.9 Exercises ... 1147

56 SOCKETS: INTRODUCTION 1149

56.1 Overview ... 1150
56.2 Creating a Socket: *socket()* ... 1153
56.3 Binding a Socket to an Address: *bind()* ... 1153
56.4 Generic Socket Address Structures: *struct sockaddr* ... 1154
56.5 Stream Sockets ... 1155
56.5.1 Listening for Incoming Connections: *listen()* ... 1156
56.5.2 Accepting a Connection: *accept()* ... 1157
56.5.3 Connecting to a Peer Socket: *connect()* ... 1158
56.5.4 I/O on Stream Sockets ... 1159
56.5.5 Connection Termination: *close()* ... 1159
56.6 Datagram Sockets ... 1159
56.6.1 Exchanging Datagrams: *recvfrom()* and *sendto()* ... 1160
56.6.2 Using *connect()* with Datagram Sockets ... 1162
56.7 Summary ... 1162

57 SOCKETS: UNIX DOMAIN 1165

57.1 UNIX Domain Socket Addresses: *struct sockaddr_un* ... 1165
57.2 Stream Sockets in the UNIX Domain ... 1167
57.3 Datagram Sockets in the UNIX Domain ... 1171
57.4 UNIX Domain Socket Permissions ... 1174
57.5 Creating a Connected Socket Pair: *socketpair()* ... 1174
57.6 The Linux Abstract Socket Namespace ... 1175
57.7 Summary ... 1176
57.8 Exercises ... 1177

58 SOCKETS: FUNDAMENTALS OF TCP/IP NETWORKS 1179

58.1 Internets ... 1179
58.2 Networking Protocols and Layers ... 1180
58.3 The Data-Link Layer ... 1182
58.4 The Network Layer: IP ... 1184
58.5 IP Addresses ... 1186
58.6 The Transport Layer ... 1188
58.6.1 Port Numbers ... 1188
58.6.2 User Datagram Protocol (UDP) ... 1189
58.6.3 Transmission Control Protocol (TCP) ... 1190
58.7 Requests for Comments (RFCs) ... 1193
58.8 Summary ... 1195

TLPI_FINAL_RRD.pdf 26 8/5/2010 10:46:57 AM

59 SOCKETS: INTERNET DOMAINS 1197

59.1 Internet Domain Sockets ... 1197
59.2 Network Byte Order ... 1198
59.3 Data Representation ... 1199
59.4 Internet Socket Addresses ... 1202
59.5 Overview of Host and Service Conversion Functions ... 1204
59.6 The *inet_pton()* and *inet_ntop()* Functions ... 1206
59.7 Client-Server Example (Datagram Sockets) ... 1207
59.8 Domain Name System (DNS) ... 1209
59.9 The `/etc/services` File ... 1212
59.10 Protocol-Independent Host and Service Conversion ... 1213
59.10.1 The *getaddrinfo()* Function ... 1213
59.10.2 Freeing *addrinfo* Lists: *freeaddrinfo()* ... 1217
59.10.3 Diagnosing Errors: *gai_strerror()* ... 1217
59.10.4 The *getnameinfo()* Function ... 1218
59.11 Client-Server Example (Stream Sockets) ... 1219
59.12 An Internet Domain Sockets Library ... 1225
59.13 Obsolete APIs for Host and Service Conversions ... 1230
59.13.1 The *inet_aton()* and *inet_ntoa()* Functions ... 1230
59.13.2 The *gethostbyname()* and *gethostbyaddr()* Functions ... 1231
59.13.3 The *getservbyname()* and *getservbyport()* Functions ... 1234
59.14 UNIX Versus Internet Domain Sockets ... 1235
59.15 Further Information ... 1235
59.16 Summary ... 1236
59.17 Exercises ... 1236

60 SOCKETS: SERVER DESIGN 1239

60.1 Iterative and Concurrent Servers ... 1239
60.2 An Iterative UDP *echo* Server ... 1240
60.3 A Concurrent TCP *echo* Server ... 1243
60.4 Other Concurrent Server Designs ... 1245
60.5 The *inetd* (Internet Superserver) Daemon ... 1247
60.6 Summary ... 1252
60.7 Exercises ... 1252

61 SOCKETS: ADVANCED TOPICS 1253

61.1 Partial Reads and Writes on Stream Sockets ... 1254
61.2 The *shutdown()* System Call ... 1256
61.3 Socket-Specific I/O System Calls: *recv()* and *send()* ... 1259
61.4 The *sendfile()* System Call ... 1260
61.5 Retrieving Socket Addresses ... 1263
61.6 A Closer Look at TCP ... 1266
61.6.1 Format of a TCP Segment ... 1266
61.6.2 TCP Sequence Numbers and Acknowledgements ... 1268
61.6.3 TCP State Machine and State Transition Diagram ... 1269
61.6.4 TCP Connection Establishment ... 1270
61.6.5 TCP Connection Termination ... 1272
61.6.6 Calling *shutdown()* on a TCP Socket ... 1273
61.6.7 The TIME_WAIT State ... 1274
61.7 Monitoring Sockets: *netstat* ... 1275
61.8 Using *tcpdump* to Monitor TCP Traffic ... 1276
61.9 Socket Options ... 1278
61.10 The `SO_REUSEADDR` Socket Option ... 1279
61.11 Inheritance of Flags and Options Across *accept()* ... 1281

TLPI_FINAL_RRD.pdf 27 8/5/2010 10:46:57 AM

61.12 TCP Versus UDP 1282
61.13 Advanced Features 1283
61.13.1 Out-of-Band Data 1283
61.13.2 The *sendmsg()* and *recvmsg()* System Calls 1284
61.13.3 Passing File Descriptors 1284
61.13.4 Receiving Sender Credentials 1284
61.13.5 Sequenced-Packet Sockets 1285
61.13.6 SCTP and DCCP Transport-Layer Protocols 1285
61.14 Summary 1286
61.15 Exercises 1287

62 TERMINALS 1289

62.1 Overview 1290
62.2 Retrieving and Modifying Terminal Attributes 1291
62.3 The *stty* Command 1294
62.4 Terminal Special Characters 1296
62.5 Terminal Flags 1301
62.6 Terminal I/O Modes 1307
62.6.1 Canonical Mode 1307
62.6.2 Noncanonical Mode 1307
62.6.3 Cooked, Cbreak, and Raw Modes 1309
62.7 Terminal Line Speed (Bit Rate) 1316
62.8 Terminal Line Control 1317
62.9 Terminal Window Size 1319
62.10 Terminal Identification 1321
62.11 Summary 1322
62.12 Exercises 1323

63 ALTERNATIVE I/O MODELS 1325

63.1 Overview 1325
63.1.1 Level-Triggered and Edge-Triggered Notification 1329
63.1.2 Employing Nonblocking I/O with Alternative I/O Models 1330
63.2 I/O Multiplexing 1330
63.2.1 The *select()* System Call 1331
63.2.2 The *poll()* System Call 1337
63.2.3 When Is a File Descriptor Ready? 1341
63.2.4 Comparison of *select()* and *poll()* 1344
63.2.5 Problems with *select()* and *poll()* 1346
63.3 Signal-Driven I/O 1346
63.3.1 When Is "I/O Possible" Signaled? 1351
63.3.2 Refining the Use of Signal-Driven I/O 1352
63.4 The *epoll* API 1355
63.4.1 Creating an *epoll* Instance: *epoll_create()* 1356
63.4.2 Modifying the *epoll* Interest List: *epoll_ctl()* 1356
63.4.3 Waiting for Events: *epoll_wait()* 1358
63.4.4 A Closer Look at *epoll* Semantics 1363
63.4.5 Performance of *epoll* Versus I/O Multiplexing 1365
63.4.6 Edge-Triggered Notification 1366
63.5 Waiting on Signals and File Descriptors 1368
63.5.1 The *pselect()* System Call 1369
63.5.2 The Self-Pipe Trick 1370
63.6 Summary 1373
63.7 Exercises 1374

TLPI_FINAL_RRD.pdf 28 8/5/2010 10:46:57 AM

64 PSEUDOTERMINALS 1375
64.1 Overview ... 1375
64.2 UNIX 98 Pseudoterminals ... 1380
64.2.1 Opening an Unused Master: *posix_openpt()* ... 1380
64.2.2 Changing Slave Ownership and Permissions: *grantpt()* ... 1381
64.2.3 Unlocking the Slave: *unlockpt()* ... 1382
64.2.4 Obtaining the Name of the Slave: *ptsname()* ... 1382
64.3 Opening a Master: *ptyMasterOpen()* ... 1383
64.4 Connecting Processes with a Pseudoterminal: *ptyFork()* ... 1385
64.5 Pseudoterminal I/O ... 1388
64.6 Implementing *script(1)* ... 1390
64.7 Terminal Attributes and Window Size ... 1394
64.8 BSD Pseudoterminals ... 1395
64.9 Summary ... 1397
64.10 Exercises ... 1398

A TRACING SYSTEM CALLS 1401

B PARSING COMMAND-LINE OPTIONS 1405

C CASTING THE NULL POINTER 1413

D KERNEL CONFIGURATION 1417

E FURTHER SOURCES OF INFORMATION 1419

F SOLUTIONS TO SELECTED EXERCISES 1425

BIBLIOGRAPHY 1437

INDEX 1447

TLPI_FINAL_RRD.pdf 29 8/5/2010 10:46:57 AM