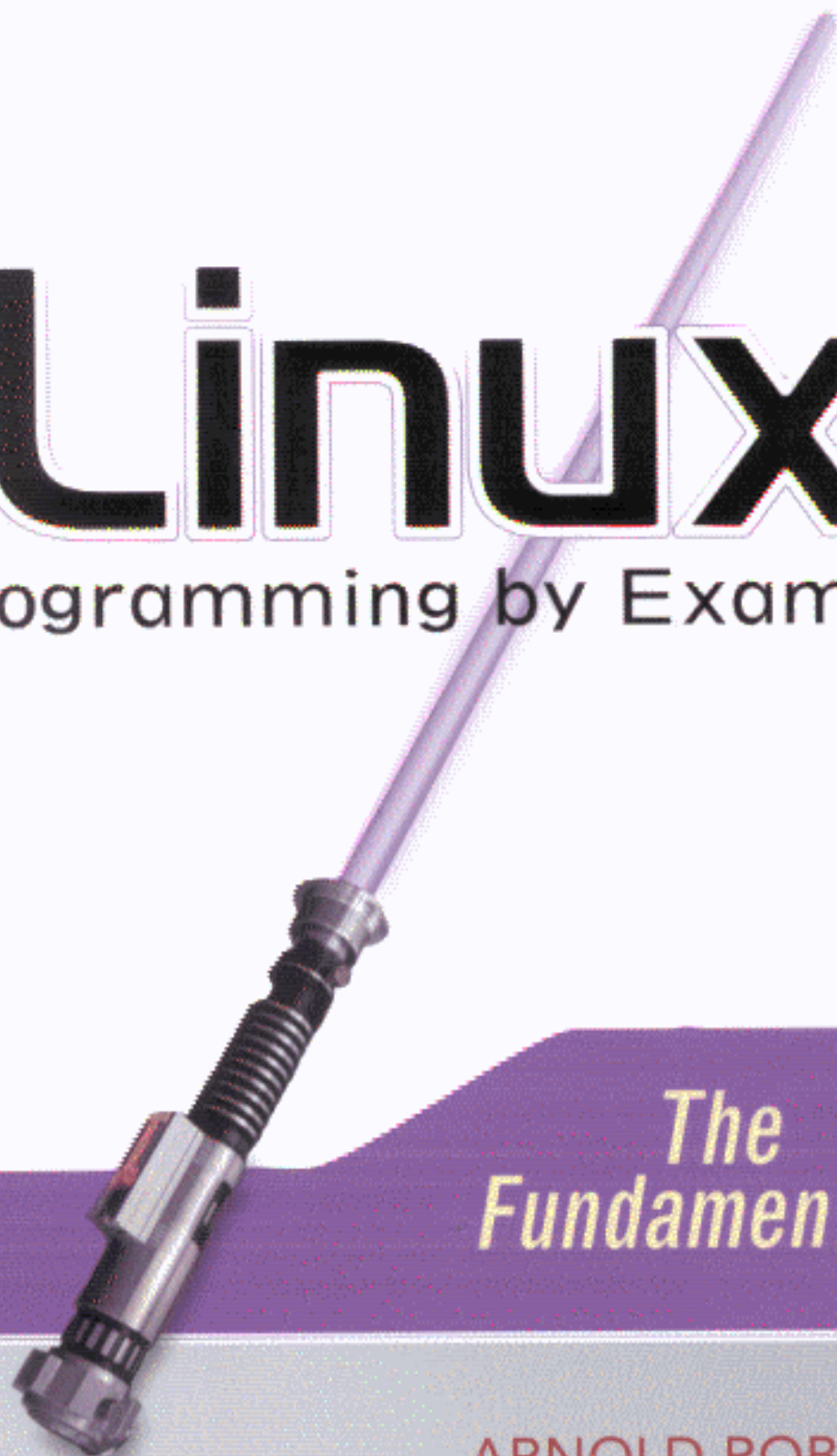


Linux[®]

Programming by Example



*The
Fundamentals*

ARNOLD ROBBINS

Contents

Preface	xvii
PART I Files and Users	1
Chapter 1 Introduction	3
1.1 The Linux/Unix File Model	4
1.1.1 Files and Permissions	4
1.1.2 Directories and Filenames	6
1.1.3 Executable Files	7
1.1.4 Devices	9
1.2 The Linux/Unix Process Model	10
1.2.1 Pipes: Hooking Processes Together	12
1.3 Standard C vs. Original C	12
1.4 Why GNU Programs Are Better	14
1.4.1 Program Design	15
1.4.2 Program Behavior	16
1.4.3 C Code Programming	16
1.4.4 Things That Make a GNU Program Better	17
1.4.5 Parting Thoughts about the “GNU Coding Standards”	19
1.5 Portability Revisited	19
1.6 Suggested Reading	20
1.7 Summary	21
Exercises	22
Chapter 2 Arguments, Options, and the Environment	23
2.1 Option and Argument Conventions	24
2.1.1 POSIX Conventions	25
2.1.2 GNU Long Options	27

2.2	Basic Command-Line Processing	28
2.2.1	The V7 echo Program	29
2.3	Option Parsing: <code>getopt()</code> and <code>getopt_long()</code>	30
2.3.1	Single-Letter Options	30
2.3.2	GNU <code>getopt()</code> and Option Ordering	33
2.3.3	Long Options	34
2.3.3.1	Long Options Table	34
2.3.3.2	Long Options, POSIX Style	37
2.3.3.3	<code>getopt_long()</code> Return Value Summary	37
2.3.3.4	GNU <code>getopt()</code> or <code>getopt_long()</code> in User Programs	39
2.4	The Environment	40
2.4.1	Environment Management Functions	41
2.4.2	The Entire Environment: <code>environ</code>	43
2.4.3	GNU <code>env</code>	44
2.5	Summary	49
	Exercises	50
Chapter 3	User-Level Memory Management	51
3.1	Linux/Unix Address Space	52
3.2	Memory Allocation	56
3.2.1	Library Calls: <code>malloc()</code> , <code>calloc()</code> , <code>realloc()</code> , <code>free()</code>	56
3.2.1.1	Examining C Language Details	57
3.2.1.2	Initially Allocating Memory: <code>malloc()</code>	58
3.2.1.3	Releasing Memory: <code>free()</code>	60
3.2.1.4	Changing Size: <code>realloc()</code>	62
3.2.1.5	Allocating and Zero-filling: <code>calloc()</code>	65
3.2.1.6	Summarizing from the <i>GNU Coding Standards</i>	66
3.2.1.7	Using Private Allocators	67
3.2.1.8	Example: Reading Arbitrarily Long Lines	67
3.2.1.9	GLIBC Only: Reading Entire Lines: <code>getline()</code> and <code>getdelim()</code> ..	73
3.2.2	String Copying: <code>strdup()</code>	74
3.2.3	System Calls: <code>brk()</code> and <code>sbrk()</code>	75
3.2.4	Lazy Programmer Calls: <code>alloca()</code>	76
3.2.5	Address Space Examination	78
3.3	Summary	80
	Exercises	81

Chapter 4	Files and File I/O	83
4.1	Introducing the Linux/Unix I/O Model	84
4.2	Presenting a Basic Program Structure	84
4.3	Determining What Went Wrong	86
4.3.1	Values for <code>errno</code>	87
4.3.2	Error Message Style	90
4.4	Doing Input and Output	91
4.4.1	Understanding File Descriptors	92
4.4.2	Opening and Closing Files	93
4.4.2.1	Mapping <code>FILE *</code> Variables to File Descriptors	95
4.4.2.2	Closing All Open Files	96
4.4.3	Reading and Writing	96
4.4.4	Example: Unix <code>cat</code>	99
4.5	Random Access: Moving Around within a File	102
4.6	Creating Files	106
4.6.1	Specifying Initial File Permissions	106
4.6.2	Creating Files with <code>creat()</code>	109
4.6.3	Revisiting <code>open()</code>	110
4.7	Forcing Data to Disk	113
4.8	Setting File Length	114
4.9	Summary	115
	Exercises	115
Chapter 5	Directories and File Metadata	117
5.1	Considering Directory Contents	118
5.1.1	Definitions	118
5.1.2	Directory Contents	120
5.1.3	Hard Links	122
5.1.3.1	The GNU <code>link</code> Program	123
5.1.3.2	Dot and Dot-Dot	125
5.1.4	File Renaming	125
5.1.5	File Removal	126
5.1.5.1	Removing Open Files	127
5.1.5.2	Using ISO C: <code>remove()</code>	127
5.1.6	Symbolic Links	128

5.2	Creating and Removing Directories	130
5.3	Reading Directories	132
5.3.1	Basic Directory Reading	133
5.3.1.1	Portability Considerations	136
5.3.1.2	Linux and BSD Directory Entries	137
5.3.2	BSD Directory Positioning Functions	138
5.4	Obtaining Information about Files	139
5.4.1	Linux File Types	139
5.4.2	Retrieving File Information	141
5.4.3	Linux Only: Specifying Higher-Precision File Times	143
5.4.4	Determining File Type	144
5.4.4.1	Device Information	147
5.4.4.2	The V7 cat Revisited	150
5.4.5	Working with Symbolic Links	151
5.5	Changing Ownership, Permission, and Modification Times	155
5.5.1	Changing File Ownership: <code>chown()</code> , <code>fchown()</code> , and <code>lchown()</code>	155
5.5.2	Changing Permissions: <code>chmod()</code> and <code>fchmod()</code>	156
5.5.3	Changing Timestamps: <code>utime()</code>	157
5.5.3.1	Faking <code>utime(file, NULL)</code>	159
5.5.4	Using <code>fchown()</code> and <code>fchmod()</code> for Security	161
5.6	Summary	162
	Exercises	163
Chapter 6	General Library Interfaces — Part 1	165
6.1	Times and Dates	166
6.1.1	Retrieving the Current Time: <code>time()</code> and <code>difftime()</code>	167
6.1.2	Breaking Down Times: <code>gmtime()</code> and <code>localtime()</code>	168
6.1.3	Formatting Dates and Times	170
6.1.3.1	Simple Time Formatting: <code>asctime()</code> and <code>ctime()</code>	170
6.1.3.2	Complex Time Formatting: <code>strftime()</code>	171
6.1.4	Converting a Broken-Down Time to a <code>time_t</code>	176
6.1.5	Getting Time-Zone Information	178
6.1.5.1	BSD Systems Gotcha: <code>timezone()</code> , Not <code>timezone</code>	179
6.2	Sorting and Searching Functions	181
6.2.1	Sorting: <code>qsort()</code>	181

6.2.1.1	Example: Sorting Employees	183
6.2.1.2	Example: Sorting Directory Contents	188
6.2.2	Binary Searching: <code>bsearch()</code>	191
6.3	User and Group Names	195
6.3.1	User Database	196
6.3.2	Group Database	199
6.4	Terminals: <code>isatty()</code>	202
6.5	Suggested Reading	203
6.6	Summary	203
	Exercises	205
Chapter 7	Putting It All Together: <code>ls</code>	207
7.1	V7 <code>ls</code> Options	208
7.2	V7 <code>ls</code> Code	209
7.3	Summary	225
	Exercises	226
Chapter 8	Filesystems and Directory Walks	227
8.1	Mounting and Unmounting Filesystems	228
8.1.1	Reviewing the Background	228
8.1.2	Looking at Different Filesystem Types	232
8.1.3	Mounting Filesystems: <code>mount</code>	236
8.1.4	Unmounting Filesystems: <code>umount</code>	237
8.2	Files for Filesystem Administration	238
8.2.1	Using Mount Options	239
8.2.2	Working with Mounted Filesystems: <code>getmntent()</code>	241
8.3	Retrieving Per-Filesystem Information	244
8.3.1	POSIX Style: <code>statvfs()</code> and <code>fstatvfs()</code>	244
8.3.2	Linux Style: <code>statfs()</code> and <code>fstatfs()</code>	252
8.4	Moving Around in the File Hierarchy	256
8.4.1	Changing Directory: <code>chdir()</code> and <code>fchdir()</code>	256
8.4.2	Getting the Current Directory: <code>getcwd()</code>	258
8.4.3	Walking a Hierarchy: <code>nftw()</code>	260
8.4.3.1	The <code>nftw()</code> Interface	261
8.4.3.2	The <code>nftw()</code> Callback Function	263

8.5	Walking a File Tree: GNU <code>du</code>	269
8.6	Changing the Root Directory: <code>chroot()</code>	276
8.7	Summary	277
	Exercises	278

PART II Processes, IPC, and Internationalization 281

Chapter 9	Process Management and Pipes	283
9.1	Process Creation and Management	284
9.1.1	Creating a Process: <code>fork()</code>	284
9.1.1.1	After the <code>fork()</code> : Shared and Distinct Attributes	285
9.1.1.2	File Descriptor Sharing	286
9.1.1.3	File Descriptor Sharing and <code>close()</code>	288
9.1.2	Identifying a Process: <code>getpid()</code> and <code>getppid()</code>	289
9.1.3	Setting Process Priority: <code>nice()</code>	291
9.1.3.1	POSIX vs. Reality	293
9.1.4	Starting New Programs: The <code>exec()</code> Family	293
9.1.4.1	The <code>execve()</code> System Call	294
9.1.4.2	Wrapper Functions: <code>execl()</code> et al.	295
9.1.4.3	Program Names and <code>argv[0]</code>	297
9.1.4.4	Attributes Inherited across <code>exec()</code>	298
9.1.5	Terminating a Process	300
9.1.5.1	Defining Process Exit Status	300
9.1.5.2	Returning from <code>main()</code>	301
9.1.5.3	Exiting Functions	302
9.1.6	Recovering a Child's Exit Status	305
9.1.6.1	Using POSIX Functions: <code>wait()</code> and <code>waitpid()</code>	306
9.1.6.2	Using BSD Functions: <code>wait3()</code> and <code>wait4()</code>	310
9.2	Process Groups	312
9.2.1	Job Control Overview	312
9.2.2	Process Group Identification: <code>getpgrp()</code> and <code>getpgid()</code>	314
9.2.3	Process Group Setting: <code>setpgid()</code> and <code>setpgrp()</code>	314
9.3	Basic Interprocess Communication: Pipes and FIFOs	315
9.3.1	Pipes	315
9.3.1.1	Creating Pipes	316
9.3.1.2	Pipe Buffering	318

9.3.2	FIFOs	319
9.4	File Descriptor Management	320
9.4.1	Duplicating Open Files: <code>dup()</code> and <code>dup2()</code>	321
9.4.2	Creating Nonlinear Pipelines: <code>/dev/fd/xx</code>	326
9.4.3	Managing File Attributes: <code>fcntl()</code>	328
9.4.3.1	The Close-on-exec Flag	329
9.4.3.2	File Descriptor Duplication	331
9.4.3.3	Manipulation of File Status Flags and Access Modes	332
9.4.3.4	Nonblocking I/O for Pipes and FIFOs	333
9.4.3.5	<code>fcntl()</code> Summary	336
9.5	Example: Two-Way Pipes in <code>gawk</code>	337
9.6	Suggested Reading	341
9.7	Summary	342
	Exercises	344
Chapter 10	Signals	347
10.1	Introduction	348
10.2	Signal Actions	348
10.3	Standard C Signals: <code>signal()</code> and <code>raise()</code>	349
10.3.1	The <code>signal()</code> Function	349
10.3.2	Sending Signals Programmatically: <code>raise()</code>	353
10.4	Signal Handlers in Action	353
10.4.1	Traditional Systems	353
10.4.2	BSD and GNU/Linux	356
10.4.3	Ignoring Signals	356
10.4.4	Restartable System Calls	357
10.4.4.1	Example: GNU Coreutils <code>safe_read()</code> and <code>safe_write()</code>	359
10.4.4.2	GLIBC Only: <code>TEMP_FAILURE_RETRY()</code>	360
10.4.5	Race Conditions and <code>sig_atomic_t</code> (ISO C)	361
10.4.6	Additional Caveats	363
10.4.7	Our Story So Far, Episode I	363
10.5	The System V Release 3 Signal APIs: <code>sigset()</code> et al.	365
10.6	POSIX Signals	367
10.6.1	Uncovering the Problem	367
10.6.2	Signal Sets: <code>sigset_t</code> and Related Functions	368

10.6.3	Managing the Signal Mask: <code>sigprocmask()</code> et al.	369
10.6.4	Catching Signals: <code>sigaction()</code>	370
10.6.5	Retrieving Pending Signals: <code>sigpending()</code>	375
10.6.6	Making Functions Interruptible: <code>siginterrupt()</code>	376
10.6.7	Sending Signals: <code>kill()</code> and <code>killpg()</code>	376
10.6.8	Our Story So Far, Episode II	378
10.7	Signals for Interprocess Communication	379
10.8	Important Special-Purpose Signals	382
10.8.1	Alarm Clocks: <code>sleep()</code> , <code>alarm()</code> , and <code>SIGALRM</code>	382
10.8.1.1	Harder but with More Control: <code>alarm()</code> and <code>SIGALRM</code>	382
10.8.1.2	Simple and Easy: <code>sleep()</code>	383
10.8.2	Job Control Signals	383
10.8.3	Parental Supervision: Three Different Strategies	385
10.8.3.1	Poor Parenting: Ignoring Children Completely	385
10.8.3.2	Permissive Parenting: Supervising Minimally	386
10.8.3.3	Strict Parental Control	393
10.9	Signals Across <code>fork()</code> and <code>exec()</code>	398
10.10	Summary	399
	Exercises	401
Chapter 11	Permissions and User and Group ID Numbers	403
11.1	Checking Permissions	404
11.1.1	Real and Effective IDs	405
11.1.2	Setuid and Setgid Bits	406
11.2	Retrieving User and Group IDs	407
11.3	Checking As the Real User: <code>access()</code>	410
11.4	Checking as the Effective User: <code>euidaccess()</code> (GLIBC)	412
11.5	Setting Extra Permission Bits for Directories	412
11.5.1	Default Group for New Files and Directories	412
11.5.2	Directories and the Sticky Bit	414
11.6	Setting Real and Effective IDs	415
11.6.1	Changing the Group Set	416
11.6.2	Changing the Real and Effective IDs	416
11.6.3	Using the Setuid and Setgid Bits	419
11.7	Working with All Three IDs: <code>getresuid()</code> and <code>setresuid()</code> (Linux)	421

11.8 Crossing a Security Minefield: Setuid root	422
11.9 Suggested Reading	423
11.10 Summary	424
Exercises	426
Chapter 12 General Library Interfaces — Part 2	427
12.1 Assertion Statements: <code>assert()</code>	428
12.2 Low-Level Memory: The <code>memXXX()</code> Functions	432
12.2.1 Setting Memory: <code>memset()</code>	432
12.2.2 Copying Memory: <code>memcpy()</code> , <code>memmove()</code> , and <code>memccpy()</code>	433
12.2.3 Comparing Memory Blocks: <code>memcmp()</code>	434
12.2.4 Searching for a Byte Value: <code>memchr()</code>	435
12.3 Temporary Files	436
12.3.1 Generating Temporary Filenames (Bad)	437
12.3.2 Creating and Opening Temporary Files (Good)	441
12.3.3 Using the <code>TMPDIR</code> Environment Variable	443
12.4 Committing Suicide: <code>abort()</code>	445
12.5 Nonlocal Gotos	446
12.5.1 Using Standard Functions: <code>setjmp()</code> and <code>longjmp()</code>	447
12.5.2 Handling Signal Masks: <code>sigsetjmp()</code> and <code>siglongjmp()</code>	449
12.5.3 Observing Important Caveats	450
12.6 Pseudorandom Numbers	454
12.6.1 Standard C: <code>rand()</code> and <code>srand()</code>	455
12.6.2 POSIX Functions: <code>random()</code> and <code>srandom()</code>	457
12.6.3 The <code>/dev/random</code> and <code>/dev/urandom</code> Special Files	460
12.7 Metacharacter Expansions	461
12.7.1 Simple Pattern Matching: <code>fnmatch()</code>	462
12.7.2 Filename Expansion: <code>glob()</code> and <code>globfree()</code>	464
12.7.3 Shell Word Expansion: <code>wordexp()</code> and <code>wordfree()</code>	470
12.8 Regular Expressions	471
12.9 Suggested Reading	480
12.10 Summary	481
Exercises	482

Chapter 13 Internationalization and Localization	485
13.1 Introduction	486
13.2 Locales and the C Library	487
13.2.1 Locale Categories and Environment Variables	487
13.2.2 Setting the Locale: <code>setlocale()</code>	489
13.2.3 String Collation: <code>strcoll()</code> and <code>strxfrm()</code>	490
13.2.4 Low-Level Numeric and Monetary Formatting: <code>localeconv()</code>	494
13.2.5 High-Level Numeric and Monetary Formatting: <code>strfmon()</code> and <code>printf()</code>	498
13.2.6 Example: Formatting Numeric Values in <code>gawk</code>	501
13.2.7 Formatting Date and Time Values: <code>ctime()</code> and <code>strftime()</code>	503
13.2.8 Other Locale Information: <code>nl_langinfo()</code>	504
13.3 Dynamic Translation of Program Messages	507
13.3.1 Setting the Text Domain: <code>textdomain()</code>	507
13.3.2 Translating Messages: <code>gettext()</code>	508
13.3.3 Working with Plurals: <code>ngettext()</code>	509
13.3.4 Making <code>gettext()</code> Easy to Use	510
13.3.4.1 Portable Programs: <code>"gettext.h"</code>	511
13.3.4.2 GLIBC Only: <code><libintl.h></code>	513
13.3.5 Rearranging Word Order with <code>printf()</code>	514
13.3.6 Testing Translations in a Private Directory	515
13.3.7 Preparing Internationalized Programs	516
13.3.8 Creating Translations	517
13.4 Can You Spell That for Me, Please?	521
13.4.1 Wide Characters	523
13.4.2 Multibyte Character Encodings	523
13.4.3 Languages	524
13.4.4 Conclusion	525
13.5 Suggested Reading	526
13.6 Summary	526
Exercises	527
Chapter 14 Extended Interfaces	529
14.1 Allocating Aligned Memory: <code>posix_memalign()</code> and <code>memalign()</code>	530
14.2 Locking Files	531

14.2.1	File Locking Concepts	531
14.2.2	POSIX Locking: <code>fcntl()</code> and <code>lockf()</code>	533
14.2.2.1	Describing a Lock	533
14.2.2.2	Obtaining and Releasing Locks	536
14.2.2.3	Observing Locking Caveats	538
14.2.3	BSD Locking: <code>flock()</code>	539
14.2.4	Mandatory Locking	540
14.3	More Precise Times	543
14.3.1	Microsecond Times: <code>gettimeofday()</code>	544
14.3.2	Microsecond File Times: <code>utimes()</code>	545
14.3.3	Interval Timers: <code>setitimer()</code> and <code>getitimer()</code>	546
14.3.4	More Exact Pauses: <code>nanosleep()</code>	550
14.4	Advanced Searching with Binary Trees	551
14.4.1	Introduction to Binary Trees	551
14.4.2	Tree Management Functions	554
14.4.3	Tree Insertion: <code>tsearch()</code>	554
14.4.4	Tree Lookup and Use of A Returned Pointer: <code>tfind()</code> and <code>tsearch()</code>	555
14.4.5	Tree Traversal: <code>twalk()</code>	557
14.4.6	Tree Node Removal and Tree Deletion: <code>tdelete()</code> and <code>tdestroy()</code> ..	561
14.5	Summary	562
	Exercises	563

PART III Debugging and Final Project 565

Chapter 15	Debugging	567
15.1	First Things First	568
15.2	Compilation for Debugging	569
15.3	GDB Basics	570
15.3.1	Running GDB	571
15.3.2	Setting Breakpoints, Single-Stepping, and Setting Watchpoints	574
15.4	Programming for Debugging	577
15.4.1	Compile-Time Debugging Code	577
15.4.1.1	Use Debugging Macros	577
15.4.1.2	Avoid Expression Macros If Possible	580
15.4.1.3	Reorder Code If Necessary	582

15.4.1.4	Use Debugging Helper Functions	584
15.4.1.5	Avoid Unions When Possible	591
15.4.2	Runtime Debugging Code	595
15.4.2.1	Add Debugging Options and Variables	595
15.4.2.2	Use Special Environment Variables	597
15.4.2.3	Add Logging Code	601
15.4.2.4	Runtime Debugging Files	602
15.4.2.5	Add Special Hooks for Breakpoints	603
15.5	Debugging Tools	605
15.5.1	The debug Library — A Sophisticated printf()	606
15.5.2	Memory Allocation Debuggers	612
15.5.2.1	GNU/Linux mtrace	613
15.5.2.2	Electric Fence	614
15.5.2.3	Debugging Malloc: dmalloc	619
15.5.2.4	Valgrind: A Versatile Tool	623
15.5.2.5	Other Malloc Debuggers	629
15.5.3	A Modern lint	631
15.6	Software Testing	632
15.7	Debugging Rules	633
15.8	Suggested Reading	637
15.9	Summary	638
Exercises		639
Chapter 16	A Project That Ties Everything Together	641
16.1	Project Description	642
16.2	Suggested Reading	644
PART IV	Appendixes	647
Appendix A	Teach Yourself Programming in Ten Years	649
Appendix B	Caldera Ancient UNIX License	655
Appendix C	GNU General Public License	657
Index		667