

# Contents at a Glance

<b>1</b>	<b>Introduction to the Linux Kernel</b>	<b>1</b>
<b>2</b>	<b>Getting Started with the Kernel</b>	<b>11</b>
<b>3</b>	<b>Process Management</b>	<b>23</b>
<b>4</b>	<b>Process Scheduling</b>	<b>41</b>
<b>5</b>	<b>System Calls</b>	<b>69</b>
<b>6</b>	<b>Kernel Data Structures</b>	<b>85</b>
<b>7</b>	<b>Interrupts and Interrupt Handlers</b>	<b>113</b>
<b>8</b>	<b>Bottom Halves and Deferring Work</b>	<b>133</b>
<b>9</b>	<b>An Introduction to Kernel Synchronization</b>	<b>161</b>
<b>10</b>	<b>Kernel Synchronization Methods</b>	<b>175</b>
<b>11</b>	<b>Timers and Time Management</b>	<b>207</b>
<b>12</b>	<b>Memory Management</b>	<b>231</b>
<b>13</b>	<b>The Virtual Filesystem</b>	<b>261</b>
<b>14</b>	<b>The Block I/O Layer</b>	<b>289</b>
<b>15</b>	<b>The Process Address Space</b>	<b>305</b>
<b>16</b>	<b>The Page Cache and Page Writeback</b>	<b>323</b>
<b>17</b>	<b>Devices and Modules</b>	<b>337</b>
<b>18</b>	<b>Debugging</b>	<b>363</b>
<b>19</b>	<b>Portability</b>	<b>379</b>
<b>20</b>	<b>Patches, Hacking, and the Community</b>	<b>395</b>
	<b>Bibliography</b>	<b>407</b>
	<b>Index</b>	<b>411</b>

# Table of Contents

<b>1</b>	<b>Introduction to the Linux Kernel</b>	<b>1</b>
	History of Unix	1
	Along Came Linus: Introduction to Linux	3
	Overview of Operating Systems and Kernels	4
	Linux Versus Classic Unix Kernels	6
	Linux Kernel Versions	8
	The Linux Kernel Development Community	10
	Before We Begin	10
<b>2</b>	<b>Getting Started with the Kernel</b>	<b>11</b>
	Obtaining the Kernel Source	11
	Using Git	11
	Installing the Kernel Source	12
	Using Patches	12
	The Kernel Source Tree	12
	Building the Kernel	13
	Configuring the Kernel	14
	Minimizing Build Noise	15
	Spawning Multiple Build Jobs	16
	Installing the New Kernel	16
	A Beast of a Different Nature	16
	No libc or Standard Headers	17
	GNU C	18
	Inline Functions	18
	Inline Assembly	19
	Branch Annotation	19
	No Memory Protection	20
	No (Easy) Use of Floating Point	20
	Small, Fixed-Size Stack	20
	Synchronization and Concurrency	21
	Importance of Portability	21
	Conclusion	21

**3 Process Management 23**

- The Process 23
- Process Descriptor and the Task Structure 24
  - Allocating the Process Descriptor 25
  - Storing the Process Descriptor 26
  - Process State 27
  - Manipulating the Current Process State 29
  - Process Context 29
  - The Process Family Tree 29
- Process Creation 31
  - Copy-on-Write 31
  - Forking 32
  - vfork() 33
- The Linux Implementation of Threads 33
  - Creating Threads 34
  - Kernel Threads 35
- Process Termination 36
  - Removing the Process Descriptor 37
  - The Dilemma of the Parentless Task 38
  - Conclusion 40

**4 Process Scheduling 41**

- Multitasking 41
- Linux's Process Scheduler 42
- Policy 43
  - I/O-Bound Versus Processor-Bound Processes 43
  - Process Priority 44
  - Timeslice 45
  - The Scheduling Policy in Action 45
- The Linux Scheduling Algorithm 46
  - Scheduler Classes 46
  - Process Scheduling in Unix Systems 47
  - Fair Scheduling 48
- The Linux Scheduling Implementation 50
  - Time Accounting 50
    - The Scheduler Entity Structure 50
    - The Virtual Runtime 51

Process Selection	52
Picking the Next Task	53
Adding Processes to the Tree	54
Removing Processes from the Tree	56
The Scheduler Entry Point	57
Sleeping and Waking Up	58
Wait Queues	58
Waking Up	61
Preemption and Context Switching	62
User Preemption	62
Kernel Preemption	63
Real-Time Scheduling Policies	64
Scheduler-Related System Calls	65
Scheduling Policy and Priority-Related System Calls	66
Processor Affinity System Calls	66
Yielding Processor Time	66
Conclusion	67
<b>5 System Calls</b>	<b>69</b>
Communicating with the Kernel	69
APIs, POSIX, and the C Library	70
Syscalls	71
System Call Numbers	72
System Call Performance	72
System Call Handler	73
Denoting the Correct System Call	73
Parameter Passing	74
System Call Implementation	74
Implementing System Calls	74
Verifying the Parameters	75
System Call Context	78
Final Steps in Binding a System Call	79
Accessing the System Call from User-Space	81
Why Not to Implement a System Call	82
Conclusion	83

**6 Kernel Data Structures 85**

- Linked Lists 85
  - Singly and Doubly Linked Lists 85
  - Circular Linked Lists 86
  - Moving Through a Linked List 87
  - The Linux Kernel's Implementation 88
    - The Linked List Structure 88
    - Defining a Linked List 89
    - List Heads 90
  - Manipulating Linked Lists 90
    - Adding a Node to a Linked List 90
    - Deleting a Node from a Linked List 91
    - Moving and Splicing Linked List Nodes 92
  - Traversing Linked Lists 93
    - The Basic Approach 93
    - The Usable Approach 93
    - Iterating Through a List Backward 94
    - Iterating While Removing 95
    - Other Linked List Methods 96
- Queues 96
  - kfifo 97
  - Creating a Queue 97
  - Enqueuing Data 98
  - Dequeuing Data 98
  - Obtaining the Size of a Queue 98
  - Resetting and Destroying the Queue 99
  - Example Queue Usage 99
- Maps 100
  - Initializing an idr 101
  - Allocating a New UID 101
  - Looking Up a UID 102
  - Removing a UID 103
  - Destroying an idr 103
- Binary Trees 103
  - Binary Search Trees 104
  - Self-Balancing Binary Search Trees 105
    - Red-Black Trees 105
    - rbtrees 106

What Data Structure to Use, When	108
Algorithmic Complexity	109
Algorithms	109
Big-O Notation	109
Big Theta Notation	109
Time Complexity	110
Conclusion	111

## **7 Interrupts and Interrupt Handlers 113**

Interrupts	113
Interrupt Handlers	114
Top Halves Versus Bottom Halves	115
Registering an Interrupt Handler	116
Interrupt Handler Flags	116
An Interrupt Example	117
Freeing an Interrupt Handler	118
Writing an Interrupt Handler	118
Shared Handlers	119
A Real-Life Interrupt Handler	120
Interrupt Context	122
Implementing Interrupt Handlers	123
/proc/interrupts	126
Interrupt Control	127
Disabling and Enabling Interrupts	127
Disabling a Specific Interrupt Line	129
Status of the Interrupt System	130
Conclusion	131

## **8 Bottom Halves and Deferring Work 133**

Bottom Halves	134
Why Bottom Halves?	134
A World of Bottom Halves	135
The Original “Bottom Half”	135
Task Queues	135
Softirqs and Tasklets	136
Dispelling the Confusion	137

Softirqs	137
Implementing Softirqs	137
The Softirq Handler	138
Executing Softirqs	138
Using Softirqs	140
Assigning an Index	140
Registering Your Handler	141
Raising Your Softirq	141
Tasklets	142
Implementing Tasklets	142
The Tasklet Structure	142
Scheduling Tasklets	143
Using Tasklets	144
Declaring Your Tasklet	144
Writing Your Tasklet Handler	145
Scheduling Your Tasklet	145
ksoftirqd	146
The Old BH Mechanism	148
Work Queues	149
Implementing Work Queues	149
Data Structures Representing the Threads	149
Data Structures Representing the Work	150
Work Queue Implementation Summary	152
Using Work Queues	153
Creating Work	153
Your Work Queue Handler	153
Scheduling Work	153
Flushing Work	154
Creating New Work Queues	154
The Old Task Queue Mechanism	155
Which Bottom Half Should I Use?	156
Locking Between the Bottom Halves	157
Disabling Bottom Halves	157
Conclusion	159
<b>9 An Introduction to Kernel Synchronization</b>	<b>161</b>
Critical Regions and Race Conditions	162
Why Do We Need Protection?	162
The Single Variable	163

Locking	165
Causes of Concurrency	167
Knowing What to Protect	168
Deadlocks	169
Contention and Scalability	171
Conclusion	172

## **10 Kernel Synchronization Methods 175**

Atomic Operations	175
Atomic Integer Operations	176
64-Bit Atomic Operations	180
Atomic Bitwise Operations	181
Spin Locks	183
Spin Lock Methods	184
Other Spin Lock Methods	186
Spin Locks and Bottom Halves	187
Reader-Writer Spin Locks	188
Semaphores	190
Counting and Binary Semaphores	191
Creating and Initializing Semaphores	192
Using Semaphores	193
Reader-Writer Semaphores	194
Mutexes	195
Semaphores Versus Mutexes	197
Spin Locks Versus Mutexes	197
Completion Variables	197
BKL: The Big Kernel Lock	198
Sequential Locks	200
Preemption Disabling	201
Ordering and Barriers	203
Conclusion	206

## **11 Timers and Time Management 207**

Kernel Notion of Time	208
The Tick Rate: HZ	208
The Ideal HZ Value	210
Advantages with a Larger HZ	210
Disadvantages with a Larger HZ	211



Jiffies	212
Internal Representation of Jiffies	213
Jiffies Wraparound	214
User-Space and HZ	216
Hardware Clocks and Timers	216
Real-Time Clock	217
System Timer	217
The Timer Interrupt Handler	217
The Time of Day	220
Timers	222
Using Timers	222
Timer Race Conditions	224
Timer Implementation	224
Delaying Execution	225
Busy Looping	225
Small Delays	226
schedule_timeout()	227
schedule_timeout() Implementation	228
Sleeping on a Wait Queue, with a Timeout	229
Conclusion	230

## **12 Memory Management 231**

Pages	231
Zones	233
Getting Pages	235
Getting Zeroed Pages	236
Freeing Pages	237
kmalloc()	238
gfp_mask Flags	238
Action Modifiers	239
Zone Modifiers	240
Type Flags	241
kfree()	243
vmalloc()	244
Slab Layer	245
Design of the Slab Layer	246

Slab Allocator Interface	249
Allocating from the Cache	250
Example of Using the Slab Allocator	251
Statically Allocating on the Stack	252
Single-Page Kernel Stacks	252
Playing Fair on the Stack	253
High Memory Mappings	253
Permanent Mappings	254
Temporary Mappings	254
Per-CPU Allocations	255
The New percpu Interface	256
Per-CPU Data at Compile-Time	256
Per-CPU Data at Runtime	257
Reasons for Using Per-CPU Data	258
Picking an Allocation Method	259
Conclusion	260

### **13 The Virtual Filesystem 261**

Common Filesystem Interface	261
Filesystem Abstraction Layer	262
Unix Filesystems	263
VFS Objects and Their Data Structures	265
The Superblock Object	266
Superblock Operations	267
The Inode Object	270
Inode Operations	271
The Dentry Object	275
Dentry State	276
The Dentry Cache	276
Dentry Operations	278
The File Object	279
File Operations	280
Data Structures Associated with Filesystems	285
Data Structures Associated with a Process	286
Conclusion	288

**14 The Block I/O Layer 289**

- Anatomy of a Block Device 290
- Buffers and Buffer Heads 291
- The bio Structure 294
  - I/O vectors 295
  - The Old Versus the New 296
- Request Queues 297
- I/O Schedulers 297
  - The Job of an I/O Scheduler 298
  - The Linus Elevator 299
  - The Deadline I/O Scheduler 300
  - The Anticipatory I/O Scheduler 302
  - The Complete Fair Queuing I/O Scheduler 303
  - The Noop I/O Scheduler 303
  - I/O Scheduler Selection 304
  - Conclusion 304

**15 The Process Address Space 305**

- Address Spaces 305
- The Memory Descriptor 306
  - Allocating a Memory Descriptor 308
  - Destroying a Memory Descriptor 309
  - The mm\_struct and Kernel Threads 309
- Virtual Memory Areas 309
  - VMA Flags 311
  - VMA Operations 312
  - Lists and Trees of Memory Areas 313
  - Memory Areas in Real Life 314
- Manipulating Memory Areas 315
  - find\_vma() 316
  - find\_vma\_prev() 317
  - find\_vma\_intersection() 317
- mmap() and do\_mmap(): Creating an Address Interval 318
- munmap() and do\_munmap(): Removing an Address Interval 320
- Page Tables 320
- Conclusion 322

<b>16</b>	<b>The Page Cache and Page Writeback</b>	<b>323</b>
	Approaches to Caching	323
	Write Caching	324
	Cache Eviction	324
	Least Recently Used	325
	The Two-List Strategy	325
	The Linux Page Cache	326
	The address_space Object	326
	address_space Operations	328
	Radix Tree	330
	The Old Page Hash Table	330
	The Buffer Cache	330
	The Flusher Threads	331
	Laptop Mode	333
	History: bdflush, kupdated, and pdflush	333
	Avoiding Congestion with Multiple Threads	334
	Conclusion	335
<b>17</b>	<b>Devices and Modules</b>	<b>337</b>
	Device Types	337
	Modules	338
	Hello, World!	338
	Building Modules	340
	Living in the Source Tree	340
	Living Externally	342
	Installing Modules	342
	Generating Module Dependencies	342
	Loading Modules	343
	Managing Configuration Options	344
	Module Parameters	346
	Exported Symbols	348
	The Device Model	348
	Kobjects	349
	Ktypes	350
	Ksets	351
	Interrelation of Kobjects, Ktypes, and Ksets	351
	Managing and Manipulating Kobjects	352

Reference Counts	353
Incrementing and Decrementing Reference Counts	354
Krefs	354
sysfs	355
Adding and Removing kobjects from sysfs	357
Adding Files to sysfs	358
Default Attributes	358
Creating New Attributes	359
Destroying Attributes	360
sysfs Conventions	360
The Kernel Events Layer	361
Conclusion	362
<b>18 Debugging</b>	<b>363</b>
Getting Started	363
Bugs in the Kernel	364
Debugging by Printing	364
Robustness	365
Loglevels	365
The Log Buffer	366
syslogd and klogd	367
Transposing printf() and printk()	367
Oops	367
ksymoops	369
kallsyms	369
Kernel Debugging Options	370
Asserting Bugs and Dumping Information	370
Magic SysRq Key	371
The Saga of a Kernel Debugger	372
gdb	372
kgdb	373
Poking and Probing the System	373
Using UID as a Conditional	373
Using Condition Variables	374
Using Statistics	374
Rate and Occurrence Limiting Your Debugging	375

Binary Searching to Find the Culprit Change	376
Binary Searching with Git	376
When All Else Fails: The Community	377
Conclusion	378
<b>19 Portability</b>	<b>379</b>
Portable Operating Systems	379
History of Portability in Linux	380
Word Size and Data Types	381
Opaque Types	384
Special Types	384
Explicitly Sized Types	385
Signedness of Chars	386
Data Alignment	386
Avoiding Alignment Issues	387
Alignment of Nonstandard Types	387
Structure Padding	387
Byte Order	389
Time	391
Page Size	391
Processor Ordering	392
SMP, Kernel Preemption, and High Memory	393
Conclusion	393
<b>20 Patches, Hacking, and the Community</b>	<b>395</b>
The Community	395
Linux Coding Style	396
Indention	396
Switch Statements	396
Spacing	397
Braces	398
Line Length	399
Naming	400
Functions	400
Comments	400
Typedefs	401
Use Existing Routines	402

Minimize ifdefs in the Source	402
Structure Initializers	402
Fixing Up Code Ex Post Facto	403
Chain of Command	403
Submitting Bug Reports	403
Patches	404
Generating Patches	404
Generating Patches with Git	405
Submitting Patches	406
Conclusion	406

**Bibliography 407**

**Index 411**