

Index

A

abstraction

address space and, 956
 architecture and, 1117
 clock sources and, 895
 kernel and, 28
 memory mappings and, 13
 virtual address space and, 289
 virtual filesystem and, 519–520,
 708

academia, role in kernel development, 1281–1282

access control lists. *See* ACLs (access control lists)

access vector cache, SELinux, 1114–1115

accessing devices. *See* device access

ACLs (access control lists), 722–732

data structures (Ext3), 726–727
 data structures (generic), 722–724
 implementing (Ext2), 731–732
 implementing (Ext3), 726
 implementing (generic), 722
 inode initialization (Ext3), 727–729
 modifying (Ext3), 730–731
 overview of, 722
 permission-checking (Ext3), 731–732
 permission-checking (generic), 724–726
 retrieving (Ext3), 729–730
 switching between on-disk and in-memory
 representation (Ext3), 726–727

active connections, TCP, 793–794

active memory regions, registering, 186–188

active pages

determining page activity, 1057–1062
 selecting pages to be swapped out, 1029
 shrinking list of, 1068–1072

address resolution protocol (ARP), 778

address space, 955–966

AMD64 systems, 188–191
 caching and, 955–966
 data structures, 956–958
 division of, 176–181
 Ex2 operations, 610–611, 637
 I/O, 395
 maximum size of, 7–8
 operations on, 961–966
 page trees and, 958–961
 PCI bus and, 455
 privilege levels and, 8–10

addresses, netlink, 811

advanced programmable interrupt controllers (APICs)

broadcast mode and, 943
 overview of, 895

alarm system calls, timer-related, 945

algorithms

page-swapping, 1026–1027
 scheduler and, 37

aliases, module names, 495

alignment, C programming, 1202–1203

generic, 1203
 natural, 1119, 1202–1203
 overview of, 1202

_alloc_pages, buddy system, 223

allocation control, page selection, 225–231

allocation macros, buddy system

allocation macros, buddy system, 220–222

allocation masks, buddy system, 205, 216–220

allocation of physical memory, 13–16

- allocation order, 135
- alternatives to slab allocator, 258–259
- boot process and, 195–197
- bootmem allocator, 191–193
- buddy system, 14–15, 204–205, 215–216
- disabling bootmem allocator, 197
- discontiguous pages in the kernel and, 244–245
- initialization and, 195–197
- macros, 220–222
- masks, 216–220
- memory areas, 248–250
- overview of, 113
- page selection and, 225–231
- slab allocator. *See* slab allocator
- slab cache, 15
- slub allocator, 150
- swapping and page reclaim, 15–16
- types of allocators, 191

Alpha systems, 1129–1131

`already_uses`, **testing module relationships, 489–490**

AMD64 systems

- address space setup, 188–191
- architecture of, 1134–1135
- clock sources, 912
- initialization of memory management, 169
- initializing of, 194
- interrupt flow handling, 870
- memory management, 134
- registering active memory regions, 187–188
- system calls and, 834
- timers for, 897

anonymous pages

- page faults, 339
- reverse mapping and, 324–325

APICs (advanced programmable interrupt controllers)

- broadcast mode and, 943
- overview of, 895

application layer, 799–808

- creating sockets, 805–807
- data structures for sockets, 799–803
- in network reference models, 736
- overview of, 799
- receiving data, 807–808
- sending data, 808
- `socketcall` system call, 804–805
- sockets and files, 803–804

architecture

- alignment of data and, 1119
- Alpha systems, 1129–1131
- AMD64 systems, 1134–1135
- architecture-specific setup, 1226–1227
- ARM systems, 1126–1127
- bit chain manipulation, 1135–1136
- checksums, 1137
- context switches, 1137–1138
- conversion to/from byte orders (little endian or big endian), 1136–1137
- `current` macro, for finding current process, 1138–1139
- data types, 1118–1119
- ELF supported, 1253
- IA-32 systems, 1122–1124
- IA-64 systems, 1124–1126
- memory pages, 1119–1120
- Mips systems, 1131–1132
- overview of, 1117–1118
- page tables, 1137
- PowerPC systems, 1132–1133
- Sparc64 systems, 1128–1129
- string processing, 1120–1121
- summary, 1139
- system calls, 1120
- thread representation, 1122

backing stores

architecture, for initialization of memory management

- address space division, 176–181
- address space setup, 188–191
- hot-n-cold cache, 183–186
- initialization steps, 172–175
- kernel setup, 169–172
- overview of, 169
- paging, 175–176
- registering active memory regions, 186–188
- virtual address space division, 181–183

architecture-independence/dependence

- CPUs and, 13, 150
- data types, 853, 1250
- high-level initialization and, 1225
- kbuild system and, 1154
- kernel and, 1043–1044, 1117
- memory management and, 160
- page flags, 151
- page tables, 154
- swapping and, 1042
- system calls, 65, 838

arithmetic

- bit operations, 1203–1206
- pointers (`ptr`), 1200

ARM systems, 1126–1127**ARP (address resolution protocol), 778****array cache, 262****assembly, in C programming**

- inline assembler, 1194–1198
- overview of, 1180

asynchronous interrupts, 848**asynchronous reading, VFS, 574****atomic operations**

- on integers, 352–353
- overview of, 351

atomic_t data types, 352–353**atomicity, locks ensuring, 359****attributes**

- C programming, 1192–1194
- Kconfig configuration, 1151

attributes, sysfs filesystem, 693–695

- data structures, 693–694
- declaring new, 694–695

audit rules, 1099**audit trails, 1098****auditctl tool, 1098****auditing, 1097–1116**

- access vector cache, 1114–1115
- audit rules, 1099
- closing audit logs, 1110
- context allocation and, 1110–1111
- data structures, 1100
- implementing, 1100
- initializing, 1106–1107
- logging events, 1108–1109
- overview of, 1097–1098
- processing requests, 1107–1108
- records, rules, and filtering, 1104–1106
- standard hooks, 1115–1116
- starting, 1109
- summary, 1116
- system call events, 1112–1114
- system calls for, 1110
- `task_struct` data structure, 1100–1104
- writing log messages, 1109

author information, modules and, 495**automatic expirations, of mounts, 563****automatic loading, modules**

- benefits of, 506–507
- `kmod` for, 480–483, 507–508

auxiliary functions, fragmentation of memory and, 205–207**B****backing stores**

- address space pages and, 956–957
- RAM and, 989
- reading data from, 297

big endian formats

big endian formats

- conversion to/from, 1136–1137
- data types, 26
- numeric values and, 739–740

big kernel lock (BKL), 361–362

binary formats

- handlers, 81–82
- interpreting, 83
- Linux supported, 82
- modules, 19–20, 491–492
- red-black trees, 299
- rewriting binary code into absolute values, 498–499

BIOs (block-based I/O)

- buffers and, 954
- creating requests, 432–436
- handling block device transfers, 969–970
- overview of, 430–431
- submitting requests, 432

bit operations

- architecture of, 1135–1136
- in C programming, 1203–1206

BKL (big kernel lock), 361–362

block allocation, Ex2

- creating new reservations, 626–628
- handling pre-allocation, 621–626
- indirection and, 619–621
- pre-allocation mechanism, 606–608

block devices

- accessing, 397
- adding disks and partitions to system, 423–425
- BIOs and, 430–431
- buffer cache. *See* buffer cache
- cache options, 950
- core properties, 415–417
- executing requests, 437–438
- generic hard disks and partitions, 417–420
- handling block device transfers, 969–970
- I/O scheduling, 438–441
- iotctl system, 441–442
- kernel request structure for, 427–430

- opening block device files, 425–427
- operations, 420
- overview of, 412–413
- page cache and. *See* page cache
- proc filesystem and, 644
- queue plugging, 436–437
- registering, 406
- representing, 413–415
- request queues, 421–423
- submitting requests, 432–436
- types of peripheral devices, 17–18

block-based filesystems, 586

block-based I/O. *See* BIOS (block-based I/O)

blocks, 586. *See also* data blocks

boot process

- data structures, 191–192
- disabling bootmem allocator, 197
- initializing, 192–194
- interface to kernel, 195–197
- overview of, 191
- releasing initialization data, 197–199

bootmem allocator

- disabling, 197
- overview of, 191–193

`_bread` function, LRU buffer cache, 987–988

break statements, C programming, 1208–1209

bridges, connecting PCI buses, 394

broadcast mode, time management and, 943–944

buckets, timer data structures, 902–904

buddy system

- `_alloc_pages`, 223
- allocation macros, 220–222
- allocation masks, 216–220
- allocation of physical memory, 14–15
- allocator API, 215–216
- avoiding fragmentation, 201
- data structures of buddy allocator, 204–205

C programming

- global variables and auxiliary functions for
 - page mobility groups, 205–207
 - grouping pages by mobility, 201–203
 - initializing mobility-based grouping, 207–208
 - managing data structure creation, 210–213
 - `_rmqueue` helper function, 234–240
 - structure of, 199–201
 - buffer cache, 974–988**
 - data structures, 975–976
 - defined, 950
 - implementing, 974
 - independent buffers, 982
 - linking buffers and pages, 977–979
 - LRU. *See* LRU (least recently used) cache
 - operations, 976–977
 - overview of, 20
 - reading whole pages into buffers, 979–981
 - structure of, 954–955
 - summary, 988
 - writing whole pages into buffers, 981–982
 - buffer head**
 - elements, 976
 - linking pages and, 977–979
 - operations, 976–977
 - structural units of buffer cache, 954–955
 - buffers, 954–955**
 - `_builtin` functions, C programming language, 1198–1200**
 - bulk transfers, USB (Universal Serial Bus), 465–466**
 - bus number, identifying PCI devices, 455**
 - bus systems**
 - generic driver model, 449
 - I/O architecture and, 392–394
 - overview of, 448
 - PCI bus. *See* PCI (peripheral component interconnect)
 - registration procedures, 452–454
 - representation of buses, 451–452
 - representation of devices, 449–451
 - USB bus. *See* USB (Universal Serial Bus)
 - buses**
 - device control via, 396
 - peripherals connected to CPU via, 393
 - registration of, 452
 - representation of, 451–452
 - byte order**
 - conversion to/from (little endian or big endian), 1136–1137
 - data types, 26–27
- C**
- C programming, 1175–1221**
 - assembly and linking, 1180
 - attributes, 1192–1194
 - bit operations, 1203–1206
 - `break` and `continue` statements, 1208–1209
 - `_builtin` functions, 1198–1200
 - common subexpression elimination technique, 1189–1190
 - data types, 1118
 - dead code elimination technique, 1190–1192
 - doubly linked lists, 1209–1214
 - Ex2 and, 586–588
 - GCC (GNU Compiler Collection), 1175
 - generic alignment, 1203
 - hash lists, 1214
 - inline assembler, 1194–1198
 - inline functions, 1192
 - Linux making use of, 1–2
 - loop optimization, 1187–1189
 - macros, 1207–1208
 - natural alignment, 1202–1203
 - optimization, 1185
 - pointer arithmetic, 1200
 - pointer type conversions, 1201–1202
 - pre-processor tricks, 1206–1207
 - procedure calls, 1180–1185
 - radix trees, 1216–1221
 - red-black trees, 1214–1216

C programming (*continued*)

C programming (*continued*)

reference counters, 1200–1201
 simplification as optimization technique, 1185–1187
 source code to machine program process, 1176–1180
 summary, 1221

caches/caching
 block devices options, 950
 buffer cache. *See* buffer cache
 comparing page cache and buffer cache, 20
 freeing memory by reducing kernel cache, 1030
 hot-n-cold pages, 183–186
 initializing during system startup, 1228–1232
 page cache. *See* page cache
 pros/cons of, 949
 shrinking, 1092–1095
 slab cache. *See* slab cache
 swap cache. *See* swap cache

call table, system calls, 834–835

canonical addresses, 189

central control, data synchronization, 1000–1002

CFS (completely fair scheduling), 106–117
 data structures, 106–107
 handling new tasks, 116–117
 handling periodic tick, 114–115
 latency tracking, 110–111
 overview of, 38, 106
 queue manipulation, 112–113
 selecting next task, 113–114
 virtual clock, 107–110
 wake-up preemption, 115–116

character devices
 accessing, 397
 block devices compared with, 412–413
 opening device files, 409–411
 range database for, 404–405
 reading from/writing to device files, 412

registering, 405–406
 representing, 409
 standard file operations, 407–408
 types of peripheral devices, 17–18

checksums

calculating, 1137
 module methods, 512–513

chicken-and-egg problem, slab allocator, 270

child processes

resource management, 443
 task relationships, 62

chip-level hardware encapsulation, interrupt handlers, 854

classes, scheduler, 89–91

classic directory allocation, Ex2, 634

classical mutexes, 362–363

classical timers. *See* low-resolution timers

cleanup functions, modules, 492–493

clock bases, 921

clock event devices

defining, 914–916
 overview of, 896, 908

clock sources, 911–913

abstraction and, 895
 defining, 911–913
 overview of, 908
 working with, 913

clone

executing system calls and, 65
 process duplication with, 64

clone

overview of, 47
 PID (process identification) and, 54

code

code generation compiler phase, 1175–1176
 coding styles and, 1274–1276
 dead code elimination, 1190
 documenting, 1276–1277
 spinlock protection and, 354–355

cumulative acknowledgement scheme, TCP

- command chain, kernel development**
 - process and, 1269
- command-line arguments, system startup, 1227–1228**
- commits, git log displaying, 1166–1167**
- common subexpression elimination, C optimization, 1189–1190**
- communication functions, network, 808–810**
- community participation, kernel development and, 1284–1287**
- completely fair scheduling. See CFS (completely fair scheduling)**
- completions, semaphores compared with, 887–888**
- computational priorities, scheduler, 94–96**
- configuration**
 - Kconfig. *See* Kconfig
 - PCI devices, 456
 - processing configuration information, 1152–1154
 - time management, 896
 - time subsystem, 909–910
 - USB devices, 464
- congested state, setting/clearing, 1011–1012**
- congestion**
 - overview of, 1009
 - setting/clearing congested state, 1011–1012
 - thresholds, 1010–1011
 - waiting on congested queues, 1012–1013
- connection termination, TCP, 797–798**
- connections, netlink, 813–814**
- containers, grouping processes in, 48**
- context switching**
 - architecture, 1137–1138
 - audit context allocation, 1110–1111
 - Lazy FPU mode and, 105–106
 - multitasking and, 102–105
- continue statements, C programming, 1208–1209**
- control groups, scheduling and, 126**
- control transfers, USB (Universal Serial Bus), 465**
- controller hardware settings, 860–861**
- copy_process function, 68–75**
- copy-on-write (COW) technique**
 - overview of, 64–65
 - page faults, 339
- core dumps, binary functions, 83**
- core scheduler**
 - interaction with generic scheduler, 86–87
 - scheduler implementation and, 99–101
 - SMP scheduling and, 125–126
- counters**
 - atomic operations on integer counters, 352–353
 - per-CPU counters, 364–365
 - race conditions and, 348–349
- COW (copy-on-write) technique**
 - overview of, 64–65
 - page faults, 339
- CPUs**
 - interaction of address space with, 13
 - locks for controlling interprocess communication, 347
 - MMU (memory management unit), 12
 - multitasking and, 35
 - per-CPU cache, 148
 - per-CPU counters, 364–365
 - scheduler fairness and, 84–85
- CRC checksums methods, modules, 512–513**
- critical sections, IPC (inter-process communication), 349–350**
- cumulative acknowledgement scheme, TCP, 794**

current

current

- macro for finding current process, 1138–1139
- system calls, 71

D

daemons. *See* kernel threads

data, application layer, 807–808

- receiving, 807–808
- sending, 808

data, module

- organizing in memory, 499–500
- transferring, 500

data, network layer, 790–791

data block sections, 588

data blocks

- allocation. *See* block allocation, Ex2
- block groups, 587
- defined, 588
- finding, 615–616
- fragmentation, 591
- group descriptors, 597–599
- indirection and, 588–591
- overview of, 413
- removing, 636
- requesting new, 616–619
- size of, 590
- superblocks, 592–597

Data Display Debugger (DDD), 1171–1172

data integrity writeback, 999

data structures, auditing, 1100

data structures, data synchronization

- congestion, 1009
- overview of, 996
- page status data structure, 996–997
- parameters, 1000
- writeback control data structure, 998–999

data structures, device drivers

- block devices, 415–417
- character device range database, 404–405
- device database, 403–404

hard disks and partitions, 417–419

I/O schedulers, 438–439

PCI bus, 458

request structure for block devices, 427–430

data structures, ELF

data types, 1250–1251

headers, 1251

overview of, 1250

program header, 1254–1255

section header, 1255–1257

string table, 1257

symbol table, 1257–1259

data structures, Ex2, 592–608

directories and files, 601–604

group descriptors, 597–599

inodes, 599–601

in memory, 604–606

pre-allocation mechanism, 606–608

superblocks, 592–597

data structures, Ex3

Ex3, 639–642

Ext3 ACLs, 726–727

Ext3 extended attributes, 714–716

data structures, extended attributes and

ACLs

ACLs (access control lists), 722–724

VFS extended attributes, 709–710

data structures, initialization of memory

management

node and zone initialization, 163–169

overview of, 162

prerequisites, 162

system start, 162–163

data structures, IPC

message queues, 377–380

signal handling, 383–386

data structures, kernel activities

interrupts, 853–856

IRQ management, 860

wait queues, 882–883

data synchronization

data structures, memory management

- architecture-independent page flags, 151–153
- boot process, 191–192
- buddy allocator, 204–205
- definition of `struct page`, 149–151
- hot-n-cold pages, 146–148
- initialization of memory management, 162
- memory zones, 140–144
- node management, 138–140
- nodes, 213–215
- page frames, 148–149
- page tables, 154–161
- slab allocator, 266–270
- `vmalloc`, 245–246
- zone watermarks, 144–146
- zones, 208–213

data structures, modules, 489–491

data structures, network

- application layer, for sockets, 799–803
- netlink, 811
- network devices, 755–759

data structures, page and buffer cache

- address spaces, 956–958
- buffer cache, 975–976
- LRU buffer cache, 983–984

data structures, page reclaim

- cache shrinkers, 1092
- page reclaim, 1055–1057
- swap areas, 1030–1031

data structures, `proc` filesystem, 652

data structures, process management

- CFS class, 106–107
- load weights, 96
- PID (process identification), 55–59
- real-time scheduling class, 118–119
- scheduler, 86–87
- scheduling entities, 92–93
- SMP scheduling, 122–124

data structures, sysfs, 690–695

- attributes, 693–695
- directory entries, 690–693
- opening files, 698–702

data structures, system startup, 1225–1226

data structures, time management

- dynamic ticks, 934
- dynamic timers, 904–905
- high-resolution timers, 921–925
- low-resolution timers, 900–902

data structures, virtual process memory

- demand paging and, 298
- priority search trees, 302
- regions, 300–302
- reverse mapping, 323–324
- `struct mm_struct`, 298
- trees and lists, 299
- virtual process memory, 303–304

data synchronization

- central control, 1000–1002
- congestion, 1009–1013
- data structures, 996
- forced writeback, 1013–1015
- full synchronization. *See* full synchronization
- inodes, 1003
- laptop mode and, 1015–1016
- overview of, 989–991
- page status data structure, 996–997
- parameters, 1000
- `pdflush` mechanism, 991–993
- periodic flushing, 996
- starting new thread, 993
- summary, 1022
- superblock inodes, 1003–1006
- superblocks, 1002
- system calls for managing, 1016
- thread initialization, 994–995
- working with `pdflush_operation`, 995
- writeback control data structure, 998–999
- writing back single inodes, 1006–1009

data types

data types, 25–27

- access to userspace and, 27
- alignment of data, 1119
- `atomic_t`, 352–353
- byte order, 26–27
- ELF, 1250–1251
- kernel architecture and, 1118–1119
- overview of, 25
- per-CPU variables, 27
- type definitions, 26

databases

- character device ranges, 404–405
- devices, 403–404

datagrams, UDP

- netlink support for, 810
- sockets for, 744–745

DDD (Data Display Debugger), 1171–1172

dead code elimination, C optimization, 1190–1192

debugfs, 687–688

debugging kernel, 1169–1173

- DDD (Data Display Debugger), 1171–1172
- examining local kernel, 1171–1172
- GDB (Gnu debugger), 1170–1172
- KGDB, 1172–1173
- overview of, 1169–1170

defragmentation, of IP packets, 772–773

demand allocation

- page faults, 337–339
- userspace page faults, 336

demand paging

- data structures and, 298
- defined, 297
- page faults, 337–339
- userspace page faults, 336

dentry cache. See directory entry cache

dependencies

- Kconfig, 1151–1152
- modules, 476–478, 488–489

depmod tool, for dependencies, 478

destination unreachable message, UDP, 785–787

development cycles, 1269–1272. See also kernel development process

device access, 397–406

- addressing using `IOCTL`, 400–401
- character devices, block devices, and other devices, 397
- device files and, 397
- dynamic creation of device files, 399–400
- identifying device files, 397–399
- major and minor numbers for representing, 401–402
- network cards and other system devices, 401
- overview of, 397
- registering character and block devices, 403–406

device drivers

- block devices. *See* block devices
- bus system and. *See* bus systems
- character devices. *See* character devices
- device access and. *See* device access
- device filesystem and. *See* device files
- generic driver model, 449
- I/O architecture for. *See* I/O (input/output) architecture
- Makefile for, 1158–1160
- overview of, 391
- PCI driver functions, 461
- PCI driver registration, 462–463
- PCI drivers, 457
- peripheral devices and, 17–18
- registering, 454
- resource reservation and. *See* resource reservations
- summary, 471
- system startup, 1234–1237
- USB drivers, 466–468

device files

- device access and, 397
- dynamic creation of, 399–400
- Ex2, 604
- file elements in inodes, 406–407

dynamic ticks

- filesystems and, 406
- identifying, 397–399
- opening for block devices, 425–427
- opening for character devices, 409–411
- random access files and, 524
- standard file operations for block devices, 408–409
- standard file operations for character devices, 407–408
- device ID, PCI devices, 456**
- device model, representation of devices, 449**
- device number, identifying PCI devices, 455**
- device special files. See device files**
- devices**
 - accessing system devices, 401
 - block devices. *See* block devices
 - character devices. *See* character devices
 - clock event devices. *See* clock event devices
 - PCI, 456
 - registering bus devices, 453
 - representation of bus devices, 449–451
 - tick devices. *See* tick devices
 - USB, 464–465
- devices, network**
 - data structures for, 755–759
 - registering, 759–760
 - representation of, 755
- diff tool, 1163–1164**
- Dijkstra, E. W., 350**
- direct reclaim, 1029**
- directories, Ex2**
 - classic directory allocation, 634
 - creating/deleting inodes, 628–630
 - deleting inodes, 634–636
 - registering inodes, 630–634
 - representation of, 601–604
- directories, *proc***
 - general system information, 648–650
 - network information, 650–651
 - system control parameters, 651–652
- directories, sysfs**
 - entries, 690–693
 - traversing, 703
- directories, VFS**
 - directory information, 540–541
 - directory trees, 549–552
- directory entry cache**
 - cache organization, 544–545
 - operations, 545–546
 - overview of, 542
 - standard functions, 546–547
 - structure of, 542–544
- disk-based filesystems, 520**
- distributed applications, race conditions and, 349**
- do_execve function, 79–83**
- do_follow_link function, 569–570**
- do_fork**
 - copy_process function and, 68–75
 - implementing, 66–67
- do_lookup function, 568**
- documenting code, in kernel development, 1276–1277**
- domain scheduling, scheduler, 123, 126–127**
- double indirection, 590**
- doubly linked lists, C programming, 1209–1214**
- dynamic linking, ELF, 1263–1265**
- dynamic ticks, 933–943**
 - configuration options and, 896
 - data structures, 934
 - for high-resolution systems, 938–939
 - for low-resolution systems, 935
 - overview of, 933–934
 - stopping/starting periodic ticks, 939–943
 - switching to, 935–936

dynamic ticks (*continued*)

dynamic ticks (*continued*)

- tick handler for, 936–937
- updating jiffies, 937–938

dynamic timers, 902–907

- activating, 907
- data structures, 904–905
- implementing handling, 905–907
- mode of operation, 902–904
- overview of, 902

E

echo client, sockets and, 740–741

echo server, sockets and, 742–744

edge-trigger interrupts, 861–863

elevators. *See* I/O (input/output)

schedulers

ELF (Executable and Linkable Format), 1241–1265

- architectures supported by, 1253
- binary format handlers, 81–82
- binary structure of modules, 491–492
- creating layout of virtual process address space, 294–296
- data structures, 1250
- data types, 1250–1251
- dynamic linking, 1263–1265
- file types, 1252
- header, 1243–1244
- header data structure, 1251
- layout and structure of, 1241–1243
- overview of, 1241
- program header data structure, 1254–1255
- program header table, 1244–1246
- relocation entries, 1259–1263
- section header data structure, 1255–1257
- sections of, 1246–1248
- string table data structure, 1257
- string tables, 1249–1250
- summary, 1265
- symbol table, 1248–1249
- symbol table data structure, 1257–1259

end points, USB devices, 465

enhanced machine, kernel as, 2

entities, schedulable, 88

entries, *proc*

- creating/registering, 660–663
- finding, 663
- representing, 652–654

entries, *sysfs* directory, 690–693

entry tasks, interrupts, 850–852

error codes, page faults, 332

errors, searching for system errors, 1232–1233

Essential Linux Device Drivers

(Vankateswaran), 391

established state, TCP, 786–787

Ethernet frames, 746

event logs. *See* logging events

Ex2. *See* second extended filesystem (Ex2)

Ex3. *See* third extended filesystem (Ex3)

exceptions

- exception fixup, kernel page faults, 341
- interrupt types, 848

exec

- COW (copy-on-write) technique and, 64
- overview of, 6, 47

Executable and Linkable Format. *See* ELF (Executable and Linkable Format)

executable files, 81

execve, starting new programs, 79–83

exit system calls, 83

exit tasks, interrupts, 850–852

expansion buses

- device files for accessing, 397
- expansion hardware, 392
- I/O memory and, 445
- types of buses, 396

exporting symbols, modules, 493–494

Ex2. *See* second extended filesystem (Ex2)

extended attributes (*xattrs*), 707–732

- ACLs. *See* ACLs (access control lists)
- data structures (Ext3), 714–716
- data structures (VFS), 709–710

filesystems

- generic handler functions, 713–714
 - implementing in Ext2, 721–722
 - interface to VFS, 708
 - listing in Ext3, 720–721
 - overview of, 707–708
 - retrieving in Ext3, 716–719
 - setting in Ext3, 719–720
 - summary, 732
 - system calls, 710–712
 - extended filesystems, 585**
 - second. *See* second extended filesystem (Ex2)
 - third. *See* third extended filesystem (Ex3)
- F**
- fallback lists, nodes, 138**
 - family relationships, task, 62–63**
 - fast path, TCP connections, 795**
 - fast userspace mutexes (futexes), 357**
 - FAT file system, inodes and, 528**
 - fault* mechanism, VFS, 576–578
 - field/comparator/value pairs, in auditing, 1099**
 - file descriptors**
 - defined, 521
 - VFS, 532–536
 - file model, VFS, 521**
 - file operations, 565–572**
 - block devices, 408–409
 - character devices, 407–408
 - `do_follow_link`, 569–570
 - `do_lookup`, 568–569
 - file representation and, 525–526
 - finding inodes, 565–567
 - opening, 570–571
 - overview of, 537–540, 565
 - reading/writing, 571–572
 - sockets, 803–804
 - file pointers, 524**
 - file-based mapping, 324–325**
 - files**
 - device. *See* device files
 - `diff` for comparing versions of, 1163
 - ELF types, 1252
 - full synchronization, 1019–1021
 - `git log` tracking development history of, 1167
 - LXR cross-referencing tool for finding, 1163
 - LXR cross-referencing tool for viewing, 1162
 - operations for accessing sockets, 803–804
 - `patch` as collection of diffs on, 1164
 - as a universal interface, 524–525
 - files, Ex2, 601–604**
 - creating/deleting inodes, 628–630
 - deleting inodes, 634–636
 - operations, 610–611
 - registering inodes, 634–636
 - size of, 590
 - files, *proc* filesystem**
 - operations, 679–680
 - processing, 668–671
 - files, *sysfs***
 - opening, 698–702
 - read/write operations, 702–703
 - files, VFS**
 - increasing initial limits, 536
 - opening, 570–571
 - operations, 537–540, 565
 - read/write operations, 571–572
 - representation of, 525–526
 - standard functions, 572–573
 - filesystems**
 - `debugfs`, 687–688
 - device files and, 406
 - devices. *See* device files
 - Ex2. *See* second extended filesystem (Ex2)
 - Ex3. *See* third extended filesystem (Ex3)
 - extended attributes. *See* extended attributes (`xattrs`)
 - kernel supported, 519
 - libfs for writing, 684–686
 - LRU (least recently used) cache and, 988

filesystems (*continued*)

filesystems (*continued*)

- overview of, 18, 644
- `proc`. *See* `proc` filesystem
- pseudo filesystems, 563–564, 689
- registering, 548–549
- sequential file interface, 680
- simple, 680
- `sysfs`. *See* `sysfs` filesystem
- system calls for managing, 828
- types of, 520
- virtual. *See* VFS (virtual filesystem)
- without persistent storage, 643–644
- writing sequential file handlers, 681–684
- fill bytes, 263–264**
- filters, audit, 1106**
- filter/value pairs, in auditing, 1099**
- fine-grained locking, 365–366**
- finite state machine, 788**
- Firewire (IEEE1394), 392, 393**
- first-fit allocator, 191**
- fixed mapping, address space division, 177, 179–181**
- flags**
 - page flags, 151–153
 - process-specific, 70
- flow handling, 860–864**
 - calling flow handler routines, 870–872
 - controller hardware settings, 860–861
 - edge-trigger interrupts, 861–863
 - interrupts, 861–864
 - interrupts handlers, 854
 - level-triggered interrupts, 863–864
 - overview of, 860–864
- flushing**
 - mechanism for, 990
 - pages, 989
 - periodic, 996
- forced writeback, 1013–1015**
 - `fork`
 - COW (copy-on-write) technique and, 64–65
 - executing system calls and, 65
 - overview of, 6, 47

- PID (process identification) and, 54
- process duplication with, 63–64
- scheduler and, 102
- forwarding packets, network layer, 774–775**
- fragmentation, network layer packets, 776**
- fragmentation, of memory**
 - avoiding, 201
 - data structure of buddy allocator and, 204–205
 - Ex2, 584, 591
 - global variables and auxiliary functions and, 205–207
 - grouping pages by mobility, 201–203
 - initializing mobility-based grouping, 207–208
 - memory management and, 15
 - virtual movable zone, 208–209
- frames**
 - Ethernet, 746
 - TCP/IP reference model, 735
- freeing memory, 250–251**
- freetext search, LXR cross-referencing tool, 1163**
- `fs_struct`, 540–541
- full synchronization. *See also* data synchronization**
 - individual files, 1019–1021
 - inodes, 1018–1019
 - memory mappings, 1021–1022
 - overview of, 1016–1018
- function number, identifying PCI devices, 455**
- functions, PID manipulation, 59–61**
- futexes (fast userspace mutexes), 357**

G

- GCC (GNU Compiler Collection), 1175**
- GCC Internals, 1175**
- GDB (GNU debugger), 1170–1172**
- general module information, 494–496**
- General Public License (GNU), 473–474**

high-resolution timers

generic alignment, C programming, 1203

generic read routine, VFS, 573–574

generic scheduler, 86–87

generic time subsystem. *See* time subsystem

genksym tool, 512–513

get free pages. *See* GFP (get free pages)

_getblk function, 985–987

GFP (get free pages)

allocation macros, 220–222

allocation masks, 216–220

reserving pages, 222–223

GiB, units of measurements, 7

Git tool, 1165–1169

exporting complete repository with,
1168–1169

incorporating modifications with,
1167–1168

overview of, 1165–1166

tracking development history with,
1166–1167

global clocks, 909

Global IDs, types of PIDs, 55

global variables, fragmentation of memory and, 205–207

GNU (General Public License), 473–474

GNU Compiler Collection (GCC), 1175

GNU debugger (GDB), 1170–1172

GNU project, 1

gotos, scheduler and, 84

grandchild processes, task relationships, 62

grandparent processes, task relationships, 62

group and domain scheduling, 126–127

group descriptors

defined, 588

Ex2, 597–599

group leader, thread groups and, 54

group scheduling

priority scheduling, 88

schedulable entities and, 126

groups, timer data structures, 902–904

H

handler functions

extended attributes (xattrs), 713–714

system calls, 830–832

handles (atomic), Ex3, 639

hard disks

adding to system, 425

generic, 417–420

hard links, 522

hard real-time processes, process priorities, 36

hardware interrupts

IRQs, 849–850

overview of, 847

hardware IRQs, 849–850

hash lists, C programming, 1214

header, ELF

data structure for, 1251

elements of, 1243–1244

heap

binary format handlers, 82

managing, 327–329

heavy-weight processes. *See* UNIX processes

helper functions

for page selection, 223–225

_rmqueue helper function, 234–240

High Precision Event Timer (HPET), 897

high-level initialization, system startup, 1225

high-level ISRs, 854

highmem capacity, memory mappings, 256

highmem pages, 134, 430

high-resolution timers, 920–933

data structures, 921–925

dynamic ticks and, 938–939

implementing in high-resolution mode,
926–929

implementing in low-resolution mode,
929–931

kernel, 16

high-resolution timers (*continued*)

high-resolution timers (*continued*)

- overview of, 920–921
- periodic tick emulation, 931–932
- setting, 925–926
- switching to, 932–933
- types of timers, 894

high-speed interfaces, for packet reception, 763–765

hooks, auditing, 1115–1116

hooks, netfilter

- activating, 783
- functions, 779–781
- table of, 781–783

host adapters, USB and, 465

host-to-network layer, in network reference models, 735

hot-n-cold pages

- initialization of cache for, 183–186
- overview of, 146–148
- refilling cache, 277–279

hotplugging

- modules and, 19, 508–511
- overview of, 18–20

hotspots, locks and, 366

HPET (High Precision Event Timer), 897

hypertext, LXR cross-referencing tool, 1161–1163

HZ frequencies, timers and, 897

I

IA-32 systems

- architecture of, 1122–1124
- clock sources, 912
- initialization of memory management and, 169
- initializing, 193–194
- interrupt flow handling, 871–872
- memory management, 134
- registering active memory regions, 187
- setting up architecture, 172–175
- system calls and, 833

system startup, 1224–1225

timers for, 897

IA-64 systems, 1124–1126

identifier search, LXR cross-referencing tool, 1162

identifiers, process. *See* PID (process identification)

IEEE (International Organization of Electrical Engineers)

- Ethernet standards, 746
- IEEE1394 (Firewire), 392, 393

implementation strategies, OSs (operating systems), 3

inactive pages

- determining page activity, 1057–1062
- reclaiming, 1072–1074
- selecting pages to be swapped out, 1029

independent buffers, 982

indirection

- block allocation and, 619–621
- double indirection, 589–590
- Ex2, 588–591
- finding data blocks, 615–616
- reading/generating indirection blocks, 615
- requesting new data blocks, 616–619
- simple indirection, 588–590
- triple indirection, 590–591

Industrial Standard Architecture (ISA), 393

init

- initialization process, 4–5
- system startup, 1233–1234

initialization

- AMD64 systems, 194
- auditing, 1106–1107
- boot process, 192–194
- high-level, 1225
- IA-32 systems, 193–194
- inodes, 727–729
- mobility-based grouping, 207–208
- modules, 492–493, 496
- nodes, 163–169
- page tables, 175–176

`proc` filesystem, 655–657
 releasing initialization data during boot
 process, 197–199
 removing initialization data, 1237–1238
 slab allocator, 270–271
 subsystem, 1225–1226
 threads, 994–995
 userspace, 1238–1239
 zones, 163–169

initialization of memory management, 161–199

address space division, 176–181
 address space setup on AMD64, 188–191
 architecture overview, 169
 boot process, 191–194
 data structure set up, 162
 data structures, 191–192
 disabling bootmem allocator, 197
 hot-n-cold cache, 183–186
 interface to kernel, 195–197
 kernel setup, 169–172
 nodes and zones, 163–169
 overview of, 161–162
 paging, 175–176
 prerequisites, 162
 registering active memory regions,
 186–188
 releasing initialization data, 197–199
 setting up architecture, 172–175
 system start, 162–163
 virtual address space division, 181–183

inline assembler, C programming, 1194–1198

inline functions, C programming, 1192

in-memory representation (Ext3), ACLs, 726–727

inode bitmaps, 588

inode initialization (Ext3), ACLs, 727–729

inode tables, 588

inode writeback

single inodes, 1006–1009
 superblock inodes, 1003–1006

inodes

block devices, 416
 data synchronization, 1003
 defined, 521
 device file elements in, 406–407
 file representation and, 525–526
 full synchronization, 1018–1019
 lists, 531–532
 lookup mechanism for finding, 565–568
 operations, 529–531
 overview of, 522
`proc` filesystem, 654–655, 668
 VFS filesystem, 527–529

inodes, Ex2

allocating, 634
 classic directory allocation, 634
 creating/deleting, 628–630
 deleting, 634–636
 operations, 610–611
 Orlov allocation, 630–634
 overview of, 599–601
 registering, 630

input/output. See I/O (input/output)

inserting regions, 309–310

instruction patterns, C programming, 1179

integers

atomic operations on, 352–353
 data types, 1118

interface functions, LRU buffer cache, 984

interface index, network devices, 760

interfaces

`debugfs` programming interface, 687–688
 files as a universal interface, 524–525
 high-speed interfaces for packet reception,
 763–765
`iotctl` (input output control interface),
 400–401, 441–442
 kernel, 195–197
 netlink programming, 814–816
 parallel, 394
 POSIX. See POSIX (Portable Operating
 System Interface for UNIX)

interfaces (continued)

interfaces (continued)

SCSI, 392, 393
 sequential files, 680
 serial, 394
 USB, 464
 VFS programming interface, 523–524
 VFS xattrs, 708

International Organization for Standardization (ISO), 734–735

International Organization of Electrical Engineers (IEEE)

Ethernet standards, 746
 IEEE1394 (Firewire), 392, 393

Internet layer, 735. *See also* network layer

Internet Protocol. *See* IP (Internet Protocol)

inter-process communication. *See* IPC (inter-process communication)

inter-process interrupt (IPIs), 943

interrupt context, 9

interrupt controllers, 849

interrupt handlers

calling high-level ISR, 872–873
 entry and exit paths, 850
 flow handling, 860–864
 function representation, 859–860
 illustration of handling an interrupt, 851
 implementing handler routines, 873–874
 requirements of, 852
 types of, 854

interrupt requests. *See* IRQs (interrupt requests)

interrupt service routine (ISR). *See* interrupt handlers

interrupts, 848–875

calling flow handler routines, 870–872
 calling high-level ISR, 872–873
 controller hardware settings, 860–861
 data structures, 853–856
 entry and exit tasks, 850–852
 flow handling, 860–864
 freeing IRQs, 865–866
 handler functions, 859–860

hardware, 847

hardware IRQs, 849–850

implementing handler routines, 873–874

interacting with peripherals, 396

interrupt handlers, 852

IRQ controller abstraction, 856–859

IRQ stacks, 869

masking, 852

PCI supported, 457

proc filesystem, 650

processing, 850

registering, 866

registering IRQs, 864–865

servicing IRQs, 866–867

sharing, 849

switching between user and kernel mode
 and, 40–41

types, 848–849

USB interrupt transfers, 466

I/O (input/output) architecture, 391–396

bus systems, 392–394

device control via buses, 396

expansion hardware and, 392

interaction with peripherals, 394–396

overview of, 391–392

I/O (input/output) memory

functions for accessing I/O memory areas,
 447

managing, 445–446

resource reservation management,
 442

I/O (input/output) ports

functions for accessing, 448

resource reservations, 442, 446–448

I/O (input/output) schedulers, 438–441

data structure for, 438–439

managing request queues, 439–440

properties, 440–441

iotctl (input output control interface) system

device addressing and, 400–401

implementation of, 441–442

IP (Internet Protocol)

address format, 736
 IPv4, 769–771
 IPv6, 783–785
 packets. *See* packets, network layer
 versions of, 736

IPC (inter-process communication)

critical sections and, 349–350
 locking mechanisms. *See* locking mechanisms
 namespace for, 50
 overview of, 347
 pipes and sockets, 389–390
 race conditions and, 348–349
 semaphores for resolving, 350
 signals. *See* signals
 summary, 390
 system calls for managing, 375, 829
 System V. *See* System V

`ipc` system call, 375

IPIs (inter-process interrupt), 943**iptables, 779****IPv4, network layer and, 769–771****IPv6, network layer and, 783–785****IRQs (interrupt requests)**

defined, 850
 freeing, 865–866
 handlers in packet reception, 765–767
 hardware IRQs, 849–850
 high-speed interfaces and, 763–764
 IRQ controller abstraction, 856–859
 IRQ stacks, 869
 registering, 864–865
 servicing, 866–867
 softIRQ handler, 767–768
 status values, 855–856

ISA (Industrial Standard Architecture), 393**ISO (International Organization for Standardization), 734–735****isochronous transfers, USB, 466****ISR (interrupt service routine). *See* interrupt handlers****J****JBD (journaling block device), 639**

`jiffies`
 time bases and, 899
 time measurement in kernel, 15–16
 updating, 937–938
 working with, 900

journal mode, Ex3, 638**journaling block device (JBD), 639****journals, Ex3, 637–638****K****Kbuild**

compiling the kernel, 1154–1156
 driver and subsystem Makefiles, 1158–1160
 main Makefile, 1157–1158
 structure of Makefiles, 1156

Kconfig

attributes, 1151
 configuration options, 1148–1150
 configuration with, 1143
 dependencies, 1151–1152
 language elements, 1147
 menu specification, 1147–1148
 sample configuration file, 1143–1147

kernel, introduction to

address spaces and privilege levels, 7–11
 allocation of physical memory, 13–16
 caching, 20
 data types, 25–27
 device drivers, block, and character devices, 17–18
 filesystems, 18
 list handling, 20–22
 modules and hotplugging, 18–20
 networks, 18
 object management and reference counting, 22–25
 ongoing evolution of, 27–28
 overview of, 3–4

kernel, introduction to (*continued*)

kernel, introduction to (*continued*)

- page tables, 11–13
- processes, task switching, and scheduling, 4
- pros/cons of, 28–29
- summary, 33
- system calls, 17
- tasks of, 2–3
- timing, 16
- Unix processes, 4–7

kernel daemons. *See* kernel threads

kernel development process

- academia and, 1281–1282
- coding styles, 1274–1276
- command chain, 1269
- community processes, 1284–1287
- development cycle, 1269–1272
- documenting code, 1276–1277
- examples of improvements to Linux, 1282–1284
- online resources and, 1272–1273
- overview of, 1267–1268
- patches and, 1273
- portability, 1276
- submission and review processes, 1277–1281
- summary, 1287
- technical issues and, 1273
- tree structure and, 1268

kernel mode, 847–891

- calling flow handler routines, 870–872
- calling high-level ISR, 872–873
- completions, 887–888
- controller hardware settings, 860–861
- data structures, 853–856
- entry and exit tasks, 850–852
- flow handling, 861–864
- freeing IRQs, 865–866
- handler function representation, 859–860
- hardware IRQs, 849–850
- implementing handler routines, 873–874
- interrupt flow handling, 860

- interrupt handlers, 852
- interrupt types, 848–849
- IRQ controller abstraction, 856–859
- IRQ stacks, 869
- overview of, 847
- preemptive multitasking and, 40–41
- privilege levels and, 8–10
- processing interrupts, 850
- registering interrupts, 866
- registering IRQs, 864–865
- servicing IRQs, 866–867
- softIRQ daemon, 878–879
- software interrupts (softIRQs), 875–877
- starting softIRQ processing, 877–878
- summary, 889–891
- switching between user and kernel mode, 833, 867–869
- tasklets, 879–882
- wait queues, 882–887
- work queues, 889–891

kernel page faults, 341–343

kernel preemption

- overview of, 41
- scheduler, 127–131

kernel space

- copying data between userspace and, 344–345
- defined, 11
- divisions of virtual address space, 7–8

kernel threads, 77–79

- address spaces and, 9–10
- implementing, 78–79
- tasks performed by, 77
- types of, 77–78

kfree

- freeing memory, 259
- freeing objects, 280
- general caches and, 283
- implementing, 285

KGDB, 1172–1173

KiB, units of measurements, 7

locking mechanisms

`kmalloc`
 general caches and, 283
 implementing, 283–285
 overview of, 259
 slab allocator and, 264
`kmap`, **251–255**
`kmod`, 480–483, 507–508
`kobjects`
 generic kernel objects, 22–24
 sets of objects, 24–25
 sysfs filesystem and, 690
`kswapd`, 1053, 1087–1090
`ktime`, **910–911**
`kupdate`, **996–997**

L

LANANA (Linux assigned name and numbers authority), 398–399
language elements, Kconfig, 1147
laptop mode, data synchronization, 1015–1016
latency
 CFS class and latency tracking, 110–111
 low latency and kernel, 131–132
layer model, in kernel, 745–747
layout, ELF, 1241–1243
layout, PCI bus, 455–457
Lazy FPU mode, 105–106
lazy TLB handling
 context switching and, 103
 overview of, 78
least recently used cache. See LRU (least recently used) cache
least recently used (LRU) algorithm, 1026–1027
level-triggered interrupts, 863–864
libfs
 pseudo filesystems and, 689
 writing filesystems with, 684–686
library, kernel as, 3

licenses, module
 overview of, 495
 querying, 500–501
life-cycle processes, 38–40
lightweight auditing framework, 1098
light-weight processes. See threads
linked computers, networks, 734
linked lists, doubly linked lists, 1209–1214
linking, in C programming, 1180
linking buffers and pages, 977–979
links, between filesystem objects, 522–523
Linux assigned name and numbers authority (LANANA), 398–399
Linux Device Drivers (Corbet), 391
Linux Gerätetreiber (Quade and Kunst), 391
Linux security modules (LSM), 830
list element, 21
list handling
 overview of, 20–21
 standard functions, 21–22
list head, 21
listen state, TCP, 786–787
lists
 data structures, 299
 RCU protected, 358–359
little endian formats
 conversion to/from, 1136–1137
 data types, 26–27
 numeric values and, 739–740
load weights, computing, 96–98
loading modules
`load_module`, 497–498
 overview of, 496
 system call for, 496–497
local clocks, 909
local IDs, types of PIDs, 55
lock contention, 365–366
lock ordering, 366
locking mechanisms, 351–366
 atomic operations on integers, 352–353
 BKL (big kernel lock), 361–362
 interprocess communication and, 347

locking mechanisms (*continued*)

locking mechanisms (*continued*)

lock contention and fine-grained locking, 365–366

memory and optimization barriers, 359–361

mutexes, 362–364

overview of, 351–352

per-CPU counters, 364–365

radix trees, 961

RCU (read-copy update), 357–359

reader/writers locks, 361

semaphores, 355–357

spinlocks, 353–355

logging events

closing logs, 1110

Ex3 log records, 639

overview of, 1108–1109

starting, 1109

writing log messages, 1109

lookup mechanisms

`do_follow_link` function, 569–570

`do_lookup` function, 568

for finding inodes, 565–568

loops, C optimization, 1187–1189

low latency, scheduler, 131–132

low-resolution timers, 897–907

data structures, 900–902

dynamic ticks and, 935

dynamic timers. *See also* dynamic timers

jiffies, 900

overview of, 897

timer activation and process accounting, 897–899

types of timers, 894

LRU (least recently used) algorithm, 1026–1027

LRU (least recently used) cache

`_bread` function, 987–988

data structures, 983–984

filesystems and, 988

`_getblk` function, 985–987

interface functions, 984

isolating LRU pages, 1065–1068

mode of operation, 982–983

overview of, 982

page reclaim data structures, 1056–1057

LSM (Linux security modules), 830

lumpy reclaim technique, 1065–1068

LXR cross-referencing tool, 1161–1163

overview of, 1161–1162

working with, 1162–1163

M

MAC addresses, TCP/IP reference model, 735

macros, C programming, 1207–1208

main scheduler, 100–101

mainline kernels, 1268

maintainers, in kernel development command chai, 1269

major numbers

device representation, 401–402

opening device files and, 411

representation of character and block devices, 398

Makefiles

driver and subsystem Makefiles, 1158–1160

main Makefile, 1157–1158

malloc, 256

mappings, memory. *See* memory mappings

masking interrupts, 852

masking signals, 381

masquerading, 778

memory addresses, alignment of data and, 1119

memory and optimization barriers, in IPC, 359–361

preemption mechanisms, 360

reorder instructions, 359–360

memory management, 133–288

architecture-independent page flags, 151–153

- checking memory utilization prior to swapping, 1029
- definition of `struct page`, 149–151
- hot-n-cold pages, 146–148
- initialization. *See* initialization of memory management
- I/O memory, 445–446
- memory zones, 140–144
- node management, 138–140
- organization in NUMA/UMA model, 136–138
- overview of, 133–136
- page frames, 148–149
- page tables. *See* page tables
- physical memory. *See* physical memory
- processor cache and TLB control, 285–287
- slab allocator. *See* slab allocator
- summary, 287–288
- swap-out for acute shortage, 1090–1092
- system calls for managing, 828
- zone watermarks, 144–146
- memory mappings**
 - abstraction and, 13
 - address spaces, 312–314
 - alternatives to `vmalloc`, 250
 - creating, 314–317
 - full synchronization, 1021–1022
 - I/O memory mapping, 395
 - kernel, 251
 - mapping functions without `highmem` capacity, 256
 - nonlinear mappings, 318–322
 - persistent kernel mappings, 251–255
 - principle of, 297–298
 - reading from, 574–576
 - removing, 317–318
 - reverse mapping, 322–327
 - temporary kernel mappings, 255–256
 - virtual process memory, 314
 - `vmalloc`. *See* `vmalloc`
- memory pages, kernel architecture and, 1119–1120**
- memory reclaim, 1086–1092**
- menu specification, Kconfig, 1147–1148**
- merge window, kernel development and, 1270**
- merging regions, 308–309**
- merging upstream, 1269**
- message format, netlink, 812–813**
- message queues, System V, 376–380**
 - data structures for, 377–380
 - FIFO (first in first out) ordering, 377
 - functional principle of, 376–377
 - overview of, 376
- MiB, units of measurements, 7**
- microkernels, implementation strategies, 3**
- migration threads**
 - code flow diagram for, 124–125
 - SMP systems and, 123
- Minix system, 584**
- minor numbers**
 - device representation, 401–402
 - opening device files and, 411
 - representation of character and block devices, 398
- Mips systems, 1131–1132**
- `mke2fs` **tool, 608–610**
- MMU (memory management unit)**
 - CPUs, 12
 - virtual memory support, 290
- mobility-based grouping, 207–208**
- `modinfo` **tool, 478–479**
- `modprobe` **tool, 480–481**
- modules, 473–517**
 - adding/removing, 474–475
 - aliases, 481–482
 - automatic loading, 480–483, 507–508
 - binary structure of, 491–492
 - binary-only, 19–20
 - as compensation for disadvantages of monolithic kernels, 3
 - CRC checksums methods, 512–513
 - dependencies, 476–477
 - exporting symbols, 493–494

modules (*continued*)

modules (*continued*)

- finding section addresses, 499
- function of, 18–19
- general module information, 494–496
- hotplugging, 19, 508–511
- initialization and cleanup functions, 492–493
- inserting/deleting, 483
- `kmod` for automatic loading, 507–508
- licenses, 495
- linking into modules and the kernel, 513–515
- loading, 496–498
- manipulating data structures, 489–491
- organizing data in memory, 499–500
- overview of, 473–474
- querying module information, 478–480
- querying module license, 500–501
- relationships between, 488–489
- removing, 505–506
- representation of, 483–488
- resolving references and relocation, 501–505
- rewriting section addresses into absolute values, 498–499
- summary, 517
- system calls for loading, 496–497
- system calls for managing, 828
- transferring data, 500
- unresolved references and, 475–476
- version control, 511–512, 515–516

modutils tool collection

- `depmod` tool, 478
- `modinfo` tool, 478
- `modprobe` tool, 480–481

monolithic kernels, implementation strategies, 3

mount points, 550

mounting/unmounting

- automatic expirations of mounts, 563
- directory trees, 549–552
- Ex2 filesystem, 612–614

- mount structures, 549–552
- `proc` filesystem, 657–659
- shared subtrees and, 558–562
- `sysfs` filesystem, 695–697
- system calls for `mount` system, 556–558
- system calls for `unmount` system, 562–563

multitasking

- context switching and, 102–105
- kernel and processor and, 35
- preemptive, 37

multithreading, 76

mutexes, 362–364

- classical, 362–363
- futexes (fast userspace mutexes), 357
- overview of, 362
- RT (real-time) mutexes, 363–364
- semaphores compared with, 356

mutual exclusion, critical sections and, 349

N

named pipes

- Ex2, 604
- random access files and, 524
- VFS, 521

namespaces

- concepts, 47–49
- implementation of, 50–52
- methods for establishing, 49
- network, 747–749
- PID (process identification) and, 55–56
- Unix processes, 7
- user namespace, 53–54
- UTS namespace, 52–53
- VFS, 541–542

NAPI (new API)

- high-speed interfaces and, 764–765
- implementing old API on top of, 768
- IRQ handlers, 766–767
- packet reception at network access layer, 760–763
- poll functions, 765–766

NAT (Network address translation), 778**natural alignment**

- C programming, 1202–1203
- overview of, 1119

nefilter

- activating hook functions, 783
- hook functions, 779–781
- overview of, 778–779
- scanning hook table, 781–783

netlink, 810–816

- data structures, 811
- message format, 812–813
- overview of, 810–811
- programming interface, 814–816
- protocol family, 811–812
- protocol-specific operations, 814
- specifying addresses, 811
- tracking netlink connections, 813–814

network access layer, 754–768

- data structures for network devices, 755–759
- high-speed interfaces for packet reception, 763–765
- IRQ handlers in packet reception, 765–768
- old API and NAPI and, 768
- overview of, 754
- poll functions in packet reception, 765–766
- receiving packets, 760
- registering network devices, 759–760
- representation of network devices, 755
- sending packets, 768
- traditional method for packet reception, 760–763
- transition from network layer to, 775–776

Network address translation (NAT), 778**network byte order, numeric values and, 739****network cards, 401****network devices**

- accessing network cards, 401
- data structures for, 755–759

- registering, 759–760
- representation of, 755

network filesystems, 520–521**network information, `proc` filesystem, 650–651****network layer, 768–785**

- activating hook functions, 783
- defragmentation of packets, 772–773
- fragmentation of packets, 776
- hook functions, 779–781
- hook table, 781–783
- IPv4, 769–771
- IPv6, 783–785
- local delivery to transport layer, 773
- netfilter, 778–779
- in network reference models, 735–736
- overview of, 768–769
- packet forwarding, 774–775
- receiving packets, 771–772
- routing, 777–778
- sending packets, 775
- transition to network access layer, 775–776

networks, 733–817

- application layer. *See* application layer
- creating sockets, 738–740
- data management using socket buffers, 750–754
- datagram sockets, 744–745
- echo client, 740–741
- echo server, 742–744
- kernel communication functions, 808–810
- layer model in kernel, 745–747
- linked computers, 734
- namespaces, 747–749
- netlink mechanism. *See* netlink
- network access layer. *See* network access layer
- network layer. *See* network layer
- networking from within kernel, 808
- overview of, 733–734
- socket buffers, 749–750

networks (*continued*)

networks (*continued*)

- sockets and, 18, 738, 740
- summary, 817
- TCP/IP reference model, 734–737
- transport layer. *See* transport layer

new API. *See* NAPI (new API)

Newton-Raphson technique, 267

nodes

- creating data structures for, 213–215
- fallback lists, 138
- initialization functions, 163–169
- initializing data structures for, 209–210
- managing, 138–139
- radix trees, 959
- RAM memory divided into, 136
- state management, 139–140

nonlinear mappings

- getting, 341
- memory mappings, 318–322
- `vm_area_struct` and, 304

non-uniform memory access. *See* NUMA (non-uniform memory access)

NUMA (non-uniform memory access)

- buddy system and, 215
- machine options for memory management, 134–136
- node and zone initialization, 163–169
- overview of, 136–138
- prerequisites for initialization of memory and, 162
- zone-specific data and, 147

O

object management, 22–25

- generic kernel objects, 22–24
- operations, 22
- reference counting and, 25
- sets of objects, 24–25

objects

- time management, 911
- VFS, 547–548

objects, slab allocator

- allocating, 276–279
- freeing, 280–282
- object poisoning, 265

offsets, virtual address space and, 12

on-disk and in-memory representation

(Ext3), ACLs, 726–727

one-shot clock event devices, 896, 917

online resources, for kernel development, 1272–1273

open source licenses, Linux, 473–474

Open Systems Interconnection (OSI), 734–735

operating systems (OSs), implementation strategies, 3

operations, 565–572

- address spaces, 961–966
- block devices, 408–409, 420
- buffer cache, 976–977
- character devices, 407–408
- dentry cache, 545–546
- device files, 407
- `do_follow_link`, 569–570
- `do_lookup`, 568–569
- file representation and, 525–526
- inodes, 529–531, 565–567
- integer counters, 352–353
- netlink, 814
- not mixing classic and atomic, 352
- opening, 570–571
- overview of, 565
- overview of file operations, 537–540
- page cache, 969–970
- `proc` files, 679–680
- reading/writing, 571–572
- socket buffers, 752
- sockets, 803–804

operations, Ex2

- address space, 637
- allocating data blocks, 619–621
- allocating inodes, 630–634
- classic directory allocation, 634

creating inodes, 628–630
 creating new reservations, 626–628
 deleting inodes, 634–636
 finding data blocks, 615–616
 handling pre-allocation, 621–626
 mounting/unmounting, 612–614
 overview of, 610–611
 registering inodes, 630
 removing data blocks, 636
 requesting new data blocks, 616–619

operations, sysfs

directory traversal, 703
 opening files, 698–702
 overview of, 697–698
 read/write, 702–703

operations, VFS, 548–564

automatic expirations of mounts, 563
 mount structures, 549–552
 mount system calls, 556–558
 registering filesystems, 548–549
 shared subtrees, 558–562
 superblock management, 552–556
 unmount system calls, 562–563

optimization, C programming

common subexpression elimination
 technique, 1189–1190
 dead code elimination technique,
 1190–1192
 loop optimization, 1187–1189
 overview of, 1185
 simplification, 1185–1187

optimization, compiler phases and, 1175

**optimization and memory barriers, IPC,
359–361**

preemption mechanisms, 360
 reorder instructions, 359–360

ordered mode, Ex3, 638

Orlov allocation, 630–634

**OSI (Open Systems Interconnection),
734–735**

**OSs (operating systems), implementation
strategies, 3**

P

packet command carriers, 429

packet filtering, netfilter, 778

packet forwarding, network layer, 774–775

packet mangling, 778

packets, network access layer

high-speed interfaces for packet reception,
 763–765
 IRQ handlers in packet reception, 765–768
 old API and NAPI and, 768
 poll functions in packet reception, 765–766
 receiving, 760
 sending, 768
 traditional method for packet reception,
 760–763

packets, network layer

defragmentation of, 772–773
 forwarding, 774–775
 fragmentation of, 776
 local delivery to transport layer, 773
 receiving, 771–772
 routing, 777–778
 sending, 775
 transition to network access layer,
 775–776

PAE (page address extension), 134

page cache, 950–954, 966–974

address spaces and, 956
 allocating pages, 966–967
 defined, 950
 finding pages, 967–968
 implementing, 966
 linking buffers and pages, 977–979
 managing/finding cached pages, 951–952
 overview of, 20
 readahead, 970–974
 reading whole pages into buffers, 979–981
 structure of, 950–951
 summary, 988
 waiting on pages, 968–969

page cache (continued)

page cache (continued)

whole page operations, 969–970
writing back modified data, 952–953
writing whole pages into buffers, 981–982

page faults. *See also* swap-page faults

allocation of physical memory, 16
anonymous pages, 339
correcting userspace page faults, 336–337
COW (copy-on-write) technique, 339
defined, 64
demand allocation/paging, 337–339
handling, 289, 330–336, 1029–1030
kernel page faults, 341–343
nonlinear mappings, 341

page flags, 151–153

page frames

allocation control for page selection,
225–231
allocation of discontinuous pages in the
kernel, 244–245
architecture-independent page flags,
151–153
defined, 11
definition of `struct page` and, 149–151
freeing pages, 240–244
grouping pages by mobility, 201–203
helper functions for page selection,
223–225
hot-n-cold pages, 146–148
linking buffers and pages, 955
overview of, 148–149
page tables and, 154
removing selected pages, 231–234
reserving, 222–223
zones and, 138

page global directory (PGD), 12

page middle directory (PMD), 12

page reclaim

activating swap areas, 1036–1039
allocation of physical memory, 16
characterization of swap areas, 1031–1033
controlling scanning, 1062–1064

creating swap areas, 1035–1036
data structures, 1055–1057
data structures for swap areas, 1030–1031
designing page reclaim and swap
subsystem, 1027–1028
determining page activity, 1057–1062
extents for non-contiguous swap areas,
1033–1035
freeing memory by reducing cache, 1030
handling page faults, 1029–1030
implementing zone shrinking, 1064–1065
isolating LRU pages and lumpy reclaim,
1065–1068
managing swap areas, 1030–1039
memory reclaim, 1086–1092
memory utilization and, 1029
overview of, 1052–1055
performing, 1075–1078
questions to answer when, 1024
reclaiming inactive pages, 1072–1074
selecting pages to be swapped out, 1029
shrinking caches/caching, 1092–1095
shrinking list of active pages, 1068–1072
shrinking zones and, 1062
swap areas. *See* swap areas
swap cache. *See* swap cache
writing data back, 1051–1052

page slots, reserving in swap areas, 1046–1049

page status, data synchronization structures, 996–997

page table entry. *See* PTE (page table entry)

page tables

breaking down virtual addresses, 154–156
creating/manipulating entries, 161
data structures, 154
defined, 12
format of, 156–158
functions for analyzing page table entries,
157
initialization functions, 175–176

permissions

- mapping virtual address space to physical
 - address space, 11–13
- memory management, 1137
- overview of, 153–154
- PTE-specific entries, 158–161
- page thrashing, 1025**
- page trees. *See also* radix trees**
 - address spaces, 958–961
 - address spaces and, 958–961
- page vectors, page reclaim data structures, 1055–1056**
- pages**
 - physical. *See* page frames
 - virtual address spaces, 11
- page-swapping algorithms, 1026–1027**
- paging. *See* swapping**
- PAL (privileged architecture level)**
 - Alpha CPUs, 1129
 - system calls and, 834
- parallel interfaces, 394**
- parameters**
 - data synchronization, 1000
 - system calls for passing, 833–834
- parent processes**
 - task relationships, 62
 - tree structure for resource management, 443
- parse trees, 1175**
- parsing, compiler phases, 1175**
- partitions**
 - adding to system, 423–425
 - addressing via device files, 398
 - generic, 417–420
- passive connections, TCP, 792–793**
- patch tool, 1164–1165**
- patches**
 - development trees and, 1268
 - origin of, 1279–1281
 - overview of, 1273
 - submitting to mailing list for review, 1277–1278
- path length, indirection, 616**
- PCI (peripheral component interconnect), 454–463**
 - address space, 455
 - configuration information, 456
 - data structures, 458
 - device management, 459–461
 - driver functions, 461
 - implementing in kernel, 457
 - layout of, 455–457
 - overview of, 393, 454–455
 - registering drivers, 462–463
 - representation of buses, 458–459
- pdfflush mechanism**
 - components of, 993
 - overview of, 991–993
 - performing work with, 995
 - thread initialization, 994
- per-CPU cache. *See* hot-n-cold pages**
- per-CPU counters, 364–365**
- per-CPU variables, 27**
- periodic flushing, data synchronization, 996**
- periodic scheduler, 99–100**
- periodic ticks**
 - CFS class, 114–115
 - dynamic ticks compared with, 934
 - emulation in high-resolution timers, 931–932
 - handler for, 936–937
 - stopping/starting, 939–943
- peripheral component interconnect. *See* PCI (peripheral component interconnect)**
- peripherals**
 - buses for connecting CPUs to, 393
 - I/O memory mapping and, 395
 - I/O ports for interacting with, 393–394
 - polling and interrupts and, 395–396
 - types of peripheral devices, 17–18
- permissions**
 - ACLs (access control lists) and, 724–726
 - Ext3, 731–732
 - IPC permissions, 376
 - VFS, 578–581

persistent mappings

persistent mappings

- address space division, 177, 179
- kernel memory mappings, 251–255

persistent sockets, Ex2, 604

PGD (page global directory), 12

physical address space, 188

physical memory, 199–256

- allocation control for page selection, 225–231
- allocation macros, 220–222
- allocation masks, 216–220
- allocation of, 13–16
- allocation of discontinuous pages in the kernel, 244–245
- alternatives to `vmalloc`, 250
- avoiding fragmentation, 201
- buddy system allocator API, 215–216
- creating data structures, 210–213
- creating `vm_area`, 247–248
- data structure of buddy system, 204–205
- data structures for each node, 213–215
- data structures of `vmalloc`, 245–246
- freeing memory, 250–251
- freeing pages, 240–244
- global variables and auxiliary functions, 205–207
- grouping pages by mobility, 201–203
- helper functions for page selection, 223–225
- initializing mobility-based grouping, 207–208
- initializing zone and node data structures, 209–210
- kernel mappings, 251
- mapping functions without `highmem` capacity, 256
- mapping virtual address space to, 10–11
- memory areas, 248–250
- page tables and, 11–13
- persistent kernel mappings, 251–255
- removing selected pages, 231–234
- reserving pages, 222–223

- `_rmqueue` helper function, 234–240
- structure of buddy system, 199–201
- temporary kernel mappings, 255–256
- virtual movable zone, 208–209
- `vmalloc`, 245

physical structure, Ex2, 585–588

PID (process identification)

- fork mechanism and, 6
- functions for manipulating, 59–61
- generating unique, 61–62
- managing, 55–59
- selecting process-specific data by, 668
- types of identifiers, 54–55

pid allocator, 55

pipes, IPC (inter-process communication), 389

PIT (programmable interrupt timer)

- ADM64 systems and, 897
- overview of, 895

PMD (page middle directory), 12

pointers (`ptr`)

- arithmetic with, 1200
- RCU protected, 357–358
- type conversion, 1201–1202

policies, scheduler, 36, 88–89

poll functions, in packet reception,

765–766

polling, interacting with peripherals,

395–396

portability, kernel development and,

1276

Portable Operating System Interface for

UNIX. See POSIX (Portable Operating System Interface for UNIX)

ports. See I/O (input/output) ports

POSIX (Portable Operating System Interface for UNIX)

- real-time scheduling class supported by Linux, 117–118
- signal handling and, 386
- system calls, 17
- system calls and, 821–822

processes

PowerPC systems

- architecture of, 1132–1133
- clock sources, 912
- system calls and, 834

pre-allocation

- handling in Ex2, 621–626
- mechanism for, 606–608

preemption counters, 128**preemption mechanisms, IPC, 360****preemptive multitasking**

- defined, 37
- overview of, 40–41

preprocessing, in compiler phases, 1175**pre-processor tricks, C programming, 1206–1207****prerequisites, initialization of memory management, 162****presentation layer, in OSI model, 736****priorities, scheduler**

- computational, 94–96
- kernel representation of, 93–94
- load weights and, 96–98
- overview of, 93

priority inheritance, 363**priority inversion, 95, 363****priority scheduling, 84**

- defined, 99
- processes, 36–38, 88

priority search trees

- overview of, 302
- representing, 304–305

private mounts, 559**privilege levels, address spaces and, 8–10****privileged architecture level (PAL)**

- Alpha CPUs, 1129
- system calls and, 834

proc filesystem, 644–680

- content categories, 644–645
- creating/registering `proc` entities, 660–663
- data structures, 652
- file operations, 679–680
- finding `proc` entities, 663

general system information, 648–650

initialization functions, 655–657

inodes, 654–655

mounting, 657–659

network information, 650–651

overview of, 643–644

processing files, 668–671

process-specific data, 645–648

reading/writing information, 664–666

representation of `proc` entries, 652–654

selecting process-specific information by PID, 668

`self` directory, 666–668

`sysctls` data structures, 673–677

`sysctls` registration, 678–679

`sysctls` static tables, 677–678

system control parameters, 651–652

system controls, 671–672

task-related information, 666

using `sysctls`, 672–673

procedure calls, C programming, 1180–1185**processes**

containers for, 48

family relationships, 62–63

finding current, 1138–1139

groups, 54

identifying. *See* PID (process identification)

implementation of namespaces, 50–52

interprocess communication. *See* IPC (inter-process communication)

life-cycle of, 38–40

managing, 35–36, 826–827

namespaces, 47–49

prioritizing, 36–38

representation of, 41–47

resource limits, 46

scheduling. *See* schedulers

states, 38–40

summary, 132

system calls. *See* system calls

task relationships, 62–63

processes (*continued*)

processes (*continued*)

- task switching and scheduling, 4
- time management for, 947–948
- types of, 47
- UNIX processes, 4–7
- user namespace, 53–54
- UTS namespace, 52–53
- wait queues for putting to sleep, 883–886
- wait queues for waking, 886–887

process-specific data, *proc* filesystem

- overview of, 645–648
- selecting by PID, 668

program header, ELF

- data structure for, 1254–1255
- overview of, 1242
- table for, 1244–1246

programmable interrupt timer (PIT)

- ADM64 systems and, 897
- overview of, 895

programming interfaces

- debugfs, 687–688
- netlink, 814–816
- VFS, 523–524

programs, system calls starting new, 79–83

properties

- block device core properties, 415–417
- I/O schedulers, 440–441
- real-time scheduling class, 118

proprietary hardware, USB and, 465

protected mode, initialization of memory and, 169

protocols, netlink

- protocol family, 811–812
- protocol-specific operations, 814

pseudo filesystems, 563–564, 689

PTE (page table entry)

- creating/manipulating entries, 161
- defined, 12
- elements, 158–161
- functions for analyzing, 157

- functions for processing
 - architecture-dependent state of
 - memory, 160

ptrace, **840–846**

Q

quadratic hashing, 634

queries

- module information, 478–480
- module licenses, 500–501

queue plugging, block devices, 436–437

queues

- request queues. *See* request queues
- run queues. *See* run queues
- waiting on congested, 1012–1013
- work queues, 889–891

R

race conditions, IPC, 348–349

radix trees

- address spaces and, 958
- C programming and, 1216–1221
- elements, 960–961
- example of, 951–952
- locking, 961
- nodes, 959
- tags, 959–960

RAM. *See also* physical memory

- allocation of, 13–16, 133–134
- backing stores and, 989
- Ex2 data structures in memory, 604–606
- I/O memory and, 445
- page cache and, 950
- page reclaim. *See* page reclaim
- page tables and, 154

random access files, 524

RB (red-black) trees

- as binary search trees, 299
- C programming and, 1214–1216
- data structures, 85

registration

RCU (read-copy update), 357–359

- core API and, 357–358
- list operations, 358–359
- overview of, 357

read memory barrier, 359**readahead**

- algorithms, 413
- page cache, 970–974
- swap-page faults and, 1085–1086

read-copy update. See RCU (read-copy update)**readelf tool, 1242****readers, RCU and, 358****reader/writers locks**

- IPC locks, 361
- overview of, 351

read/write operations

- asynchronous reading, 574
- block devices, 421–423
- character devices, 412
- generic read routine, 573–574
- `proc` filesystem, 664–666
- reading from mappings, 574–576
- `sysfs` filesystem, 702–703
- VFS files, 571–572
- whole pages into buffers, 979–982

real-time (RT) mutexes, 363–364**real-time scheduling class, 117–121**

- data structures, 118–119
- operations, 119–121
- overview of, 117–118
- properties, 118

real-time timers, 946–947**receiving packets**

- high-speed interfaces for packet reception, 763–765
- IRQ handlers in packet reception, 765–768
- network access layer, 760
- network layer, 771–772
- poll functions in packet reception, 765–766
- traditional method, 760–763
- transport layer, 795–796

records, audit record formats, 1104**red zoning, slab allocator and, 265****red-black (RB) trees**

- as binary search trees, 299
- C programming and, 1214–1216
- data structures, 85

reference counters

- C programming language, 1200–1201
- object management and, 25

references

- handling unresolved references for
 - modules, 475–476
- relationships between modules, 488–489
- resolving references and relocation, 501–505
- unresolved, 514

regions

- associating virtual addresses with, 306–308
- creating, 310–312
- inserting, 309–310
- merging, 308–309
- overview of, 306
- representing, 300–302

register transfer language (RTL), 1175, 1179**registration**

- active memory regions, 186–188
- block devices, 406
- bus system, 452–454
- cache shrinkers, 1092
- character devices, 405–406
- filesystems, 548–549
- inodes (Ex2), 630–636
- interrupts, 866
- IRQs, 864–865
- network devices, 759–760
- PCI drivers, 462–463
- `proc` entries, 660–663
- `sysctls`, 678–679
- `sysfs` subsystems, 704–706
- tasklets, 880–881

Reiserfs

Reiserfs, 528

relationships, between modules, 488–489

relative addresses, relocatable files and, 475

releasing pages, 989

relocatable modules, 475

relocation entries, ELF, 1259–1263

data structures, 1261–1262

overview of, 1259–1261

relative displacement example, 1262–1263

relocation types, 1262

reorder instructions, memory and optimization barriers and, 359–360

Request for Comments (RFCs), 737

request management queues, block devices, 413–415

request queues

block devices, 421–423

hardware characteristics of, 424

I/O scheduler managing, 439–440

queue plugging, 436–437

reservations, creating in Ex2, 626–628

resource limits (rlimit), 45–47

overview of, 45–46

process-specific, 46–47

resource manager, kernel as, 2

resource reservations

I/O memory and, 445–446

I/O ports and, 446–448

overview of, 442

requesting/releasing resources, 444

tree data structure for managing, 442–444

return values, system calls, 835–837

reverse mapping, 322–327

benefits of, 289

creating, 324–325

data structures, 323–324

overview of, 322–323

swap cache and, 1040

using, 325–326

review process, kernel development and, 1277–1281

RFCs (Request for Comments), 737

RISC machines, 1129

rlimit (resource limits), 45–47

overview of, 45–46

process-specific, 46–47

.rmb(), 359

_rmqueue helper function, buddy system, 234–240

ROM, 445

routing, 777–778

routing tables, 774

RT (real-time) mutexes, 363–364

RTL (register transfer language), 1175, 1179

rules, audit, 1104–1106

run queues

defined, 85

manipulating, 112–113

overview of, 91–92

scheduling domains and, 123

running state, processes, 38–39

rwlock_t, 361

S

SBus, 393

scan operations, shrinking zones and, 1062–1064

scanning, compiler phases, 1175

sched.h, 41–43

schedulable entities, 88, 126

schedule function, 84

schedulers

CFS class. *See* CFS (completely fair scheduling)

computational priorities, 94–96

context switching and, 102–105

data structures, 86–87

elements in task structure of processes and, 87–89

enhancements, 121

entity data structure, 92–93

semaphores. *See also* mutexes

- fork mechanism and, 102
- group and domain scheduling, 126–127
- I/O schedulers, 438–441
- kernel preemption and, 127–131
- Lazy FPU mode and, 105–106
- load weights in prioritization, 96–98
- low latency and, 131–132
- main scheduler, 100–101
- overview of, 83–86
- periodic scheduler, 99–100
- priorities and, 93
- process scheduling, 4, 16
- real-time scheduling class. *See* real-time scheduling class
- representation of priorities in kernel, 93–94
- run queues, 91–92
- scheduling domains, 123, 126–127
- SMP scheduling, 121–126
- system calls for managing, 827–828
- tasks of, 36
- types of, 37–38
- scheduling classes**
 - operations performed by, 90–91
 - overview of, 87
- scheduling entities**
 - CFS (completely fair scheduling) and, 38
 - data structure of, 92–93
- SCSI (Small Computer System Interface)**
 - expansion buses, 392
 - overview of, 393
- search tags, 952**
- second chance algorithm, page-swapping, 1026**
- second extended filesystem (Ex2), 584–637**
 - ACLs (access control lists) and, 732
 - address space operations, 637
 - allocating data blocks, 619–621
 - classic directory allocation, 634
 - creating filesystem, 608–610
 - creating reservations, 626–628
 - creating/deleting inodes, 628–630
 - data structures, 592
 - data structures in memory, 604–606
 - deleting inodes, 634–636
 - directories and files, 601–604
 - extended attributes, 721–722
 - finding data blocks, 615–616
 - fragmentation and, 591
 - group descriptors, 597–599
 - handling pre-allocation, 621–626
 - indirection, 588–591
 - inodes, 599–601
 - introduction to, 583–584
 - mounting/unmounting, 612–614
 - operations, 610–611
 - Orlov allocation of inodes, 630–634
 - overview of, 584–585
 - physical structure, 585–588
 - pre-allocation mechanism, 606–608
 - registering inodes, 630
 - removing data blocks, 636
 - requesting new data blocks, 616–619
 - summary, 642
 - superblocks, 592–597
- section addresses, modules**
 - finding, 499
 - rewriting binary code into absolute values, 498–499
- section header, ELF**
 - data structure for, 1255–1257
 - overview of, 1242
 - table for, 1246–1248
- sections, ELF, 1242, 1246–1248**
- sectors, 413**
- segmentation faults, 64**
 - self **directory**, **proc filesystem**, 666–668
- semaphores. *See also* mutexes**
 - completions compared with, 887
 - kernel space, 355–357
 - overview of, 351
 - reader/writers locks, 361
 - resolving IPC problems with, 350
 - userspace. *See* semaphores, System V

semaphores, System V

semaphores, System V

- data structures, 369–375
- IPC permissions, 376
- overview of, 367–369
- system calls, 375

sending packets

- network access layer, 768
- network layer, 775
- transport layer, 796–797

sending signals, 387–388

sequential file interface

- connecting with VFS, 684
- overview of, 680
- writing sequential file handlers, 681–684

serial interfaces, 394

session layer, in OSI model, 736

sessions, process groups combined in, 54

setitimer system call, 945

shared libraries, 83

shared mappings, 301

shared memory, System V, 380–381

shared mounts, 558

shared subtrees, VFS, 558–562

shrinkers, cache

- data structures, 1092
- registering/removing, 1092
- shrinking caches with, 1093–1095

sibling processes, tree structures for resource management, 443

signals

- data structures for signal handling, 383–386
- default actions for standard, 385
- implementing signal handlers, 382–383
- overview of, 381–389
- processing signal queue, 388–389
- sending, 387–388
- system calls for implementing signal handling, 386–387
- system calls for managing signal handling, 827

simple file system, 680–689

simplification, C optimization techniques, 1185–1187

single indirect blocks, 590

slab allocator, 256–285

- allocating objects, 276–279
- alternative allocators to, 258–259
- chicken-and-egg problem, 270
- creating caches, 271–276
- data structures for implementing, 266–270
- destroying caches, 283
- fineness of cache structure and, 261–262
- fineness of slab structure and, 262–265
- freeing objects, 280–282
- general caches, 283–285
- growing the cache, 279–280
- implementing, 265
- initializing, 270–271
- memory management in kernel and, 259–261
- overview of, 256–258
- principle of, 261

slab cache

- allocation of physical memory, 15
- buffer head and, 976
- creating, 271–276
- destroying, 283
- fineness of cache structure, 261–262
- general caches, 283–285
- growing, 279–280
- processor cache and TLB control, 285–287
- shrinking, 1092

slave mounts, 558

sleeping state, processes, 38–39

slob allocator

- alternatives to slab allocator, 258
- shrinking slob cache, 1092

slow path, TCP connections, 795

slub allocator

- allocation of physical memory, 150
- alternatives to slab allocator, 258
- shrinking slub cache, 1092

sparse superblock technique

Small Computer System Interface (SCSI)

- expansion buses, 392
- overview of, 393

SMP scheduling, 121–126

- core scheduler changes, 125–126
- extensions to data structures, 122–124
- migration thread, 124–125
- overview of, 121–122

SMP systems

- system start, 163
- timers and, 898

socket buffers

- data management elements, 753–754
- operations on, 752
- overview of, 749–750
- pointers for managing protocol headers, 750–753

socketcall system call, 804–806**sockets**

- creating, 738–740
- data management using socket buffers, 750–754
- datagram sockets, 744–745
- echo client, 740–741
- echo server, 742–744
- Ex2, 604
- IPC, 389–390
- networks and, 18
- TCP and, 790
- UDP and, 786–787
- using, 740

sockets, application layer

- creating, 805–807
- data structures, 799–803
- socketcall system call, 804–806
- sockets and files, 803–804

soft links, 523**soft real-time processes, 36****softIRQs**

- completions, 887–888
- handler for, 767–768
- overview of, 847, 875–877

- softIRQ daemon, 878–879
- starting softIRQ processing, 877–878
- tasklets, 879–882
- wait queues, 882–887
- work queues, 889–891

software I/O mapping, 445**source code**

- attributes, 1151
- configuration options, 1148–1150
- DDD (Data Display Debugger), 1171–1172
- debugging and analyzing the kernel, 1169–1170
- dependencies, 1151–1152
- diff tool, 1163–1164
- examining local kernel, 1171–1172
- GDB (Gnu debugger), 1170–1172
- Git tool, 1165–1169
- kbuild for compiling the kernel, 1154–1156
- Kconfig for configuring, 1143
- KGDB, 1172–1173
- language elements of kconfig, 1147
- LXR cross-referencing tool, 1161–1163
- Makefiles component, 1156–1160
- menu specification, 1147–1148
- online resources for kernel development, 1272–1273
- organization of, 1141–1143
- overview of, 1141
- patch tool, 1164–1165
- phases in translating to machine code, 1176–1180
- processing configuration information, 1152–1154
- sample configuration file, 1143–1147
- summary, 1174
- tools for working with, 1160
- UML (User-Mode Linux), 1173–1174

source navigation, LXR cross-referencing tool, 1162**source navigation, LXR cross-referencing tool, 1162****Sparc64 systems, 1128–1129****sparse superblock technique, 587**

spinlock_t

spinlock_t, 354**spinlocks, 353–355**data structures for, and use of,
354–355

kernel preemption and, 128

overview of, 351

protecting code sections with, 353

reader/writers locks, 361

semaphores compared with, 356

standard functions, VFS, 572–581

asynchronous reading, 574

fault mechanism, 576–578

generic read routine, 573–574

overview of, 572–573

permission-checking, 578–581

reading from mappings, 574–576

starting tracing, system calls, 843**states, process, 38–40****stopping tracing, system calls, 845**

strace, 821–822, 838–840

streams, file communication and, 524**string processing, kernel architecture and,
1120–1121****string tables, ELF**

data structure for, 1257

overview of, 1249–1250

stringification, 1206struct pid, **56–57**struct file, **534–535**struct mm_struct, **298**struct module, **483–488**

struct page

definition of, 149–151

page frames and, 148–149

struct upid, **56–57****subexpressions, common subexpression
elimination, 1189–1190****submission process, kernel development
and, 1277–1281****submitting requests, block devices,
432–436****subnets, TCP/IP reference model, 736****subsystems**

data types, 1118

Makefile for, 1158–1160

sysfs, 704–706

system startup, 1225–1226

superblocks

data synchronization, 1002

defined, 588

Ex2, 592–597, 612

inodes, 1003–1006

managing, 552–556

sparse superblock technique, 587

swap areas

activating, 1036

characterization of, 1031–1033

creating, 1035–1036

data structures for, 1030–1031

extents for non-contiguous, 1033–1035

extents list for, 1038–1039

managing, 1030–1039

organization of, 1028

reading swap area characteristics,
1037–1038**swap cache**

adding new pages to, 1045–1046

allocating swap space, 1049

caching swap pages, 1050

identifying swapped-out pages, 1041–1044

overview of, 1039–1041

reserving page slots, 1046–1049

searching for a swap pages, 1050–1051

structure of, 1044–1045

swap tokens

overview of, 1079–1082

page thrashing and, 1025

swap_writepage **function, 1051–1052****swap-page faults, 1082–1086**

overview of, 1082

readahead, 1085–1086

reading the data, 1084–1085

swapping, 1082–1086

swapping pages in, 1083–1084

1330

swapped-out pages, identifying, 1041–1044
swapping

- activating swap areas, 1036–1039
- allocation of physical memory, 15
- characterization of swap areas, 1031–1033
- creating swap areas, 1035–1036
- data structures for swap areas, 1030–1031
- designing page reclaim and swap subsystem, 1027–1028
- extents for non-contiguous swap areas, 1033–1035
- freeing memory by reducing cache, 1030
- handling page faults, 1029–1030
- managing swap areas, 1030–1039
- memory utilization and, 1029
- overview of, 1023–1024
- page thrashing and, 1025
- pages, 989
- page-swapping algorithms, 1026–1027
- questions to answer when, 1024
- selecting pages to be swapped out, 1029
- swap-page faults, 1082–1086
- synchronization compared with, 990
- types of swappable pages, 1024–1025

switch_to, 104**symbol tables, ELF**

- data structure for, 1257–1259
- overview of, 1248–1249

symbolic links, 522–523, 603**symbols**

- exporting symbols, 493–494
- resolving, 502–505

synchronization of data. See data synchronization**synchronous interrupts, 848****synchronous reads, 578–581****sysctls**

- data structures, 673–677
- overview of, 671–672
- parameters, 651–652
- registering, 678–679

- static tables, 677–678
- using, 672–673

sysfs filesystem, 689–706

- attributes, 693–695
- data structures, 690
- directory entries, 690–693
- directory traversal, 703
- file and directory operations, 697–698
- mounting/unmounting, 695–697
- opening files, 698–702
- overview of, 643–644, 689–690
- populating, 704
- read/write operations, 702–703
- registering subsystems, 704–706
- summary, 706

system buses, 396**system calls, 63–83, 819–846**

- access to userspace and, 837–838
- available, 826–830
- call tables and, 834–835
- categories of, 17
- copying processes (`copy_process` function), 68–75
- COW (copy-on-write) technique and, 64–65
- data synchronization and, 1016
- `do_fork` implementation, 66–67
- executing, 65–66
- exiting processes, 79–83
- extended attributes (`xattrs`) and, 710–712
- `fork` for process duplication, 63–64
- handler functions, 830–832
- kernel architecture and, 1120
- loading modules, 496–497
- `mount` system, 556–558
- overview of, 63, 819
- parameter passing, 833–834
- `ptrace`, 840–846
- restarting, 824–826
- return values and, 835–837
- signal handling and, 386–387
- `socketcall`, 804–806
- standards supported, 823–824

system calls (*continued*)

system calls (*continued*)

- starting new programs, 79–83
- `strace`, 838–840
- structure of, 830
- summary, 846
- switching between user and kernel mode, 9
- system programming and, 820
- thread generation, 75–77
- threads and, 77–79
- timer-related, 944–947
- tracing and, 820–822, 843, 845
- `unmount`, 562–563
- VFS files, 538–539
- VFS programming interface and, 523–524
- system calls, for auditing, 1110–1116**
 - access vector cache auditing, 1114–1115
 - audit context allocation, 1110–1111
 - standard hooks, 1115–1116
 - system call events, 1112–1114
- system controls. See `sysctls`**
- system devices, accessing, 401**
- system errors, searching for, 1232–1233**
- system information**
 - `proc` filesystem, 648–650
 - system calls for managing, 829
- system management, kernel issues related to, 35–36**
- system programming, 820**
- system security, 829–830**
- system settings, 829**
- system startup, 1223–1239**
 - architecture-specific setup, 1226–1227
 - command-line arguments, 1227–1228
 - driver setup, 1234–1237
 - high-level initialization, 1225
 - IA-32 systems, 1224–1225
 - `init` thread, 1233–1234
 - initialization functions, 162–163
 - initializing central data structures and caches, 1228–1232
 - overview of, 1223
 - removing initialization data, 1237–1238

- searching for system errors, 1232–1233
- subsystem initialization, 1225–1226
- summary, 1239
- userspace initialization, 1238–1239

system trace. See tracing system calls

System V

- interprocess communication and, 347
- IPC mechanisms, 366–367
- IPC permissions, 376
- message queues, 376–380
- overview of, 366
- semaphore data structures, 369–375
- semaphore system calls, 375
- semaphore use, 367–369
- shared memory, 380–381

T

tags, radix trees, 959–960

tags, search tags, 952

task switching

- processes and, 4
- scheduler and, 87
- scheduler policy compared with, 36

`task_struct` **data structure**

- code listing, 41–43
- `exit_state` element, 44
- extension for auditing, 1100–1104
- resource limits, 45–47
- sections of, 44
- `state` element, 44

tasklets, 879–882

- executing, 881–882
- generating, 880
- overview of, 879–880
- registering, 880–881

tasks

- elements in process task structure, 87–89
- `proc` filesystem, 666
- process management and, 62–63
- relating task structures to namespaces, 59–62

time subsystem

tasks, CFS class

- handling new tasks, 116–117
- selecting next task, 113–114

TCP (Transmission Control Protocol)

- active connections, 793–794
- connection termination, 797–798
- overview of, 787–788
- passive connections, 792–793
- receiving packets, 795–796
- receiving TCP data, 790–791
- sending packets, 796–797
- TCP headers, 788–789
- three-way handshake, 791–792
- transmitting packets, 794
- UDP compared with, 744–745

TCP/IP reference model, 734–737**technical issues, kernel development and, 1273****telnet tool, 737****temporary kernel mappings, 255–256****text segment, binary format handlers, 82****TGID (thread group ID), 54****third extended filesystem (Ex3), 637–642**

- concepts, 638–639
- data structures, 639–642
- data structures for ACLs, 726–727
- implementing ACLs, 726
- inode initialization, 727–729
- introduction to, 583–584
- listing extended attributes, 720–721
- log records, handles, and transactions, 639
- modifying ACLs, 730–731
- overview of, 637–638
- permission-checking, 731–732
- retrieving ACLs, 729–730
- retrieving extended attributes, 716–719
- setting extended attributes, 719–720
- summary, 642
- switching between on-disk and in-memory representation, 727
- writeback mode, 964

thread group ID (TGID), 54**threads**

- data synchronization, 993
- initializing, 994–995
- program execution and, 6–7
- representation of, 1122
- system calls generating, 75–77

three-way handshake, TCP, 791–792**thresholds, congestion, 1010–1011****tick devices**

- overview of, 908
- time subsystem, 916–920

tickless systems, 933**ticks**

- dynamic ticks. *See* dynamic ticks
- periodic. *See* periodic ticks

time bases, 944–945**time management, 893–984**

- broadcast mode and, 943–944
 - configuration options, 896
 - dynamic ticks. *See* dynamic ticks
 - dynamic timers. *See* dynamic timers
 - generic time subsystem. *See* time subsystem
 - getting current time, 947
 - high-resolution timers. *See* high-resolution timers
 - implementing timer-related system calls, 944–947
 - kernel and, 16
 - low-resolution timers. *See* low-resolution timers
 - managing process times, 947–948
 - overview of, 893
 - representation of time, 910–911
 - summary, 948
 - system calls, 827
 - types of timers, 893–896
- time ordering, 359–360**
- time stamp counter (TSC), 912**
- time subsystem, 907–920**
- clock event devices, 914–916
 - clock sources, 911–913

time subsystem (*continued*)

time subsystem (*continued*)

configuration options, 909–910
 objects for time management, 911
 overview of, 907–909
 representation of time, 910–911
 tick devices, 916–920

time-based deferral of tasks, 893

time-outs, 895

timer wheel timers. *See* low-resolution timers

timers

dynamic. *See* dynamic timers
 high-resolution. *See* high-resolution timers
 low-resolution. *See* low-resolution timers
 setting, 925–926
 types of, 893–896

TLB (Translation Lookaside Buffer)

defined, 13
 processor cache and, 285–287

tools

DDD (Data Display Debugger),
 1171–1172
 debugging and analyzing the kernel,
 1169–1170
 diff tool, 1163–1164
 examining local kernel, 1171–1172
 GDB (Gnu debugger), 1170–1172
 Git tool, 1165–1169
 KGDB, 1172–1173
 LXR cross-referencing tool, 1161–1163
 overview of, 1160
 patch tool, 1164–1165

tracing system calls, 820–822

ptrace, 840–846
 starting tracing, 843
 stopping tracing, 845
 strace, 838–840

traditional method, receiving packets, 760–763

transactions, Ex3

overview of, 639
 saving in journal, 638

Translation Lookaside Buffer (TLB)

defined, 13
 processor cache and, 285–287

Transmission Control Protocol. *See* TCP (Transmission Control Protocol)

transmitting packets, TCP, 794

transparent proxy, 778

transport layer, 785–799

active connections, 793–794
 connection termination, 797–798
 local delivery from network layer,
 773
 overview of, 736, 785
 passive connections, 792–793
 receiving packets, 795–796
 receiving TCP data, 790–791
 sending packets, 796–797
 TCP, 787–788
 TCP headers, 788–789
 three-way handshake, 791–792
 transmitting packets, 794
 UPD, 785–787

trees

data structures, 299
 kernel development process and,
 1268
 representation of device tree, 468–471
 resource reservation management,
 442–444
 USB, 464, 468–471
 VFS directory trees, 549–552
 VFS shared subtrees, 558–562

triple indirection, 590

TSC (time stamp counter), 912

type definitions, data types, 26

U

UDP (User Datagram Protocol)

datagram sockets, 744–745
 overview of, 785–787

UID (user identification), 47–48

VFS (virtual filesystem)

UMA (uniform memory access)

- buddy system and, 215
- machine options for memory management, 134–136
- node and zone initialization, 163–169
- overview of, 136–138
- prerequisites for initialization of memory and, 162

UML (User-Mode Linux), 1173–1174

`uname` system, 47

unbindable mounts, 559**unbounded priority inversion, 363****uniform memory access. See UMA (uniform memory access)****units of measurements, conventions for, 7****Universal Serial Bus. See USB (Universal Serial Bus)****UNIX processes, 4–7**

- fork and exec mechanisms, 6
- `init` program, 4–6
- namespaces, 7
- threads, 6–7

UNIX Timesharing System (UTS) namespace

- overview of, 52–53
- running kernel an, 50

unresolved references, modules, 475–476**USB (Universal Serial Bus)**

- driver management, 466–468
- expansion buses, 392
- features and mode of operation, 463–466
- hotplugging and, 509–510
- overview of, 393, 463
- representation of device tree, 468–471

User Datagram Protocol (UDP)

- datagram sockets, 744–745
- overview of, 785–787

user identification (UID), 47–48**user mode**

- preemptive multitasking and, 40–41
- privilege levels and, 8–10

- switching between user and kernel mode, 833
- system call return values and, 835–837

user namespace, 53–54**userland**

- defined, 11
- virtual address space, 289

User-Mode Linux (UML), 1173–1174**userspace**

- copying data between kernel and, 344–345
- correcting userspace page faults, 336–337
- data type access to, 27
- defined, 11
- divisions of virtual address space, 7–8
- initialization, 1238–1239
- semaphores. *See* semaphores, System V
- system calls accessing, 837–838

UTS (UNIX Timesharing System) namespace

- overview of, 52–53
- running kernel an, 50

V**vanilla kernels, 1268****vendor ID, configuring PCI devices, 456****version control, modules**

- elementary, 495–496
- functions, 515–516
- overview of, 511–512

vfork

- executing system calls and, 65
- process duplication with, 64

VFS (virtual filesystem), 519–581

- associated files, 532–536
- asynchronous reading, 574
- automatic expirations of mounts, 563
- common file model, 521
- components of, 525
- connecting sequential files, 684
- connecting with, 689
- `dentry` cache organization, 544–545
- `dentry` operations, 545–546

VFS (virtual filesystem) (*continued*)

VFS (virtual filesystem) (*continued*)

`dentry` standard functions, 546–547
`dentry` structure, 542–544
 device files and, 406
 directory entry cache, 542
 directory information, 540–541
`do_follow_link` function, 569–570
`do_lookup` function, 568
`fault` mechanism, 576–578
 file descriptors, 532–536
 file operations, 537–540, 565
 file representation, 525–526
 files as a universal interface, 524–525
 filesystem and superblock information, 526–527
 filesystem types, 520
 generic read routine, 573–574
 inode lists, 531–532
 inode operations, 529–531
 inode structure, 527–529
 inodes generally, 522
 interface for extended attributes (`xattrs`) to, 708
 links, 522–523
 lookup mechanism for finding inodes, 565–568
 mounting/unmounting directory trees, 549–552
 namespaces, 541–542
 opening files, 570–571
 overview of, 18, 519–520
 permission-checking, 578–581
 programming interface, 523–524
 pseudo filesystems, 563–564
 reading from mappings, 574–576
 read/write operations, 571–572
 registering filesystems, 548–549
 shared subtrees, 558–562
 standard functions, 572–573
 summary, 581
 superblock information, 526–527
 superblock management, 552–556

system calls for `mount` system, 556–558
 system calls for `unmount` system, 562–563
 type of filesystems, 520
 working with VFS objects, 547–548

virtual address space

breaking down into parts, 154–156
 defined, 7
 division of, 181–183
 mapping physical memory to, 10–11
 page tables and, 11–13
 setting up address space for AMD64 systems, 188–189

virtual address space, for processes

associating virtual addresses with regions, 306–308
 creating layout, 294–296
 layout of, 290–294
 mapping, 312–314
 overview of, 290

virtual clocks

`CFS` (completely fair scheduling) operations and, 107–110
 run queues and, 85–86

virtual filesystems

`proc` filesystem. *See* `proc` filesystem
 VFS. *See* VFS (virtual filesystem)

virtual memory maps (VMMs), 190

virtual movable zone, 208–209

virtual process memory, 289–345

address spaces, mapping, 312–314
 anonymous pages, 339
 associating virtual addresses with regions, 306–308
 copying data between kernel and userspace, 344–345
 correcting page faults, 336–337
`COW` (copy-on-write) technique and, 339
 creating layout of address space, 294–296
 creating memory mappings, 314–317
 creating regions, 310–312
 data structures, 298, 303–304

zones, of memory

demand allocation/paging, 337–339
 heap management, 327–329
 inserting regions, 309–310
 kernel page faults, 341–343
 layout of address space, 290–294
 memory mappings, 297–298, 314
 merging regions, 308–309
 nonlinear mappings, 318–322, 341
 operations on regions, 306
 overview of, 4, 289–290
 page cache and, 950
 page fault handling, 330–336
 priority search trees, 302
 regions, 300–302
 removing memory mappings,
 317–318
 representing priority trees, 304–305
 reverse mapping, 322–327
 summary, 345
 trees and lists, 299
 virtual address space, 290

vm_area
 allocating memory areas,
 248–250
 creating, 247–248

vm_area_struct
 nonlinear mappings and, 304
 regions as instances of, 299–300
 representing priority trees, 304–305

vmalloc
 address space division, 177–179
 allocating memory areas, 248–250
 alternatives to, 250
 creating **vm_area**, 247–248
 data structures, 245–246
 freeing memory, 250–251

overview of, 245
 VMMs (virtual memory maps) and, 190
VMMs (virtual memory maps), 190

W

wait queues, 882–887
 congestion and, 1009
 data structures, 882–883
 putting processes to sleep, 883–886
 waking processes, 886–887

waiting state, processes, 38–39

wake-up preemption, CFS class, 115–116

work queues, 889–891

write operations. See read/write operations

writeback

allocation of physical memory, 16
 control structure, 998–999
 Ex3, 638, 964
 forced, 1013–1015
 page cache and, 952–953
 page reclaim and, 1051–1052
 single inodes, 1006–1009

writers, RCU and, 358

Z

“zombie state”, processes, 39–40

zone modifiers, 216–218

zones, of memory

calculating watermarks, 144–146
 data structures for, 209–210
 implementing shrinking, 1064–1065
 initialization functions, 163–169
 overview of, 136–138, 140–144
 shrinking, 1062
 virtual movable zone, 208–209