Contents

To Everyone iii
To Educators vi
To Students viii
Acknowledgments ix
Final Words xiii
References xiv

1 A Dialogue on the Book 1

2 Introduction to Operating Systems 3
2.1 Virtualizing The CPU 5
2.2 Virtualizing Memory 7
2.3 Concurrency 9
2.4 Persistence 11
2.5 Design Goals 13
2.6 Some History 14
2.7 Summary 19
References 20
Homework 21

I Virtualization 23

3 A Dialogue on Virtualization 25

4 The Abstraction: The Process 27
4.1 The Abstraction: A Process 28
4.2 Process API 29
4.3 Process Creation: A Little More Detail 30
4.4 Process States 31
4.5 Data Structures 33
4.6 Summary 35
References 37

Homework (Simulation) 38

5 Interlude: Process API 41
5.1 The `fork()` System Call 41
5.2 The `wait()` System Call 44
5.3 Finally, The `exec()` System Call 44
5.4 Why? Motivating The API 46
5.5 Process Control And Users 48
5.6 Useful Tools 49
5.7 Summary 50
References 52
Homework (Simulation) 53
Homework (Code) 54

6 Mechanism: Limited Direct Execution 57
6.1 Basic Technique: Limited Direct Execution 57
6.2 Problem #1: Restricted Operations 58
6.3 Problem #2: Switching Between Processes 63
6.4 Worried About Concurrency? 67
6.5 Summary 68
References 71
Homework (Measurement) 72

7 Scheduling: Introduction 73
7.1 Workload Assumptions 73
7.2 Scheduling Metrics 74
7.3 First In, First Out (FIFO) 74
7.4 Shortest Job First (SJF) 76
7.5 Shortest Time-to-Completion First (STCF) 77
7.6 A New Metric: Response Time 78
7.7 Round Robin 79
7.8 Incorporating I/O 81
7.9 No More Oracle 82
7.10 Summary 83
References 84
Homework (Simulation) 85

8 Scheduling:
The Multi-Level Feedback Queue 87
8.1 MLFQ: Basic Rules 88
8.2 Attempt #1: How To Change Priority 89
8.3 Attempt #2: The Priority Boost 92
8.4 Attempt #3: Better Accounting 93
8.5 Tuning MLFQ And Other Issues 94
8.6 MLFQ: Summary 96
References 97
Homework (Simulation) 98

9 Scheduling: Proportional Share 99
9.1 Basic Concept: Tickets Represent Your Share 99
9.2 Ticket Mechanisms 101
9.3 Implementation 102
9.4 An Example 103
9.5 How To Assign Tickets? 104
9.6 Stride Scheduling 104
9.7 The Linux Completely Fair Scheduler (CFS) 105
9.8 Summary 110
References 111
Homework (Simulation) 112

10 Multiprocessor Scheduling (Advanced) 113
10.1 Background: Multiprocessor Architecture 114
10.2 Don't Forget Synchronization 116
10.3 One Final Issue: Cache Affinity 117
10.4 Single-Queue Scheduling 118
10.5 Multi-Queue Scheduling 119
10.6 Linux Multiprocessor Schedulers 122
10.7 Summary 122
References 123
Homework (Simulation) 124

11 Summary Dialogue on CPU Virtualization 127

12 A Dialogue on Memory Virtualization 129

13 The Abstraction: Address Spaces 131
13.1 Early Systems 131
13.2 Multiprogramming and Time Sharing 131
13.3 The Address Space 133
13.4 Goals 135
13.5 Summary 136
References 138
Homework (Code) 139

14 Interlude: Memory API 141
14.1 Types of Memory 141
14.2 The `malloc()` Call 142
14.3 The `free()` Call 144
14.4 Common Errors 144
14.5 Underlying OS Support 148
14.6 Other Calls 148
14.7 Summary 149
References 150
Homework (Code) 151

15 Mechanism: Address Translation 153
15.1 Assumptions . . . 154
15.2 An Example . . . 154
15.3 Dynamic (Hardware-based) Relocation . . . 157
15.4 Hardware Support: A Summary . . . 160
15.5 Operating System Issues . . . 161
15.6 Summary . . . 163
References . . . 166
Homework (Simulation) . . . 167

16 Segmentation 169
16.1 Segmentation: Generalized Base/Bounds . . . 169
16.2 Which Segment Are We Referring To? . . . 172
16.3 What About The Stack? . . . 174
16.4 Support for Sharing . . . 175
16.5 Fine-grained vs. Coarse-grained Segmentation . . . 175
16.6 OS Support . . . 176
16.7 Summary . . . 178
References . . . 179
Homework (Simulation) . . . 180

17 Free-Space Management 181
17.1 Assumptions . . . 182
17.2 Low-level Mechanisms . . . 183
17.3 Basic Strategies . . . 191
17.4 Other Approaches . . . 193
17.5 Summary . . . 195
References . . . 197
Homework (Simulation) . . . 198

18 Paging: Introduction 199
18.1 A Simple Example And Overview . . . 199
18.2 Where Are Page Tables Stored? . . . 203
18.3 What's Actually In The Page Table? . . . 204
18.4 Paging: Also Too Slow . . . 206
18.5 A Memory Trace . . . 207
18.6 Summary . . . 210
References . . . 211
Homework (Simulation) . . . 212

19 Paging: Faster Translations (TLBs) 215
19.1 TLB Basic Algorithm . . . 216
19.2 Example: Accessing An Array . . . 217
19.3 Who Handles The TLB Miss? . . . 220
19.4 TLB Contents: What's In There? . . . 222
19.5 TLB Issue: Context Switches . . . 223
19.6 Issue: Replacement Policy . . . 225

19.7 A Real TLB Entry 225
19.8 Summary 226
References 228
Homework (Measurement) 229

20 Paging: Smaller Tables 231
20.1 Simple Solution: Bigger Pages 231
20.2 Hybrid Approach: Paging and Segments 232
20.3 Multi-level Page Tables 235
20.4 Inverted Page Tables 243
20.5 Swapping the Page Tables to Disk 243
20.6 Summary 243
References 244
Homework (Simulation) 245

21 Beyond Physical Memory: Mechanisms 247
21.1 Swap Space 248
21.2 The Present Bit 249
21.3 The Page Fault 250
21.4 What If Memory Is Full? 251
21.5 Page Fault Control Flow 252
21.6 When Replacements Really Occur 253
21.7 Summary 254
References 255
Homework (Measurement) 256

22 Beyond Physical Memory: Policies 259
22.1 Cache Management 259
22.2 The Optimal Replacement Policy 260
22.3 A Simple Policy: FIFO 262
22.4 Another Simple Policy: Random 264
22.5 Using History: LRU 265
22.6 Workload Examples 266
22.7 Implementing Historical Algorithms 269
22.8 Approximating LRU 270
22.9 Considering Dirty Pages 271
22.10 Other VM Policies 272
22.11 Thrashing 272
22.12 Summary 273
References 274
Homework (Simulation) 276

23 Complete Virtual Memory Systems 277
23.1 VAX/VMS Virtual Memory 278
23.2 The Linux Virtual Memory System 284
23.3 Summary 293
References 295

24 Summary Dialogue on Memory Virtualization 297

II Concurrency 301

25 A Dialogue on Concurrency 303

26 Concurrency: An Introduction 305
26.1 Why Use Threads? . . . 306
26.2 An Example: Thread Creation . . . 307
26.3 Why It Gets Worse: Shared Data . . . 310
26.4 The Heart Of The Problem: Uncontrolled Scheduling . . . 313
26.5 The Wish For Atomicity . . . 315
26.6 One More Problem: Waiting For Another . . . 316
26.7 Summary: Why in OS Class? . . . 317
References . . . 318
Homework (Simulation) . . . 319

27 Interlude: Thread API 321
27.1 Thread Creation . . . 321
27.2 Thread Completion . . . 322
27.3 Locks . . . 325
27.4 Condition Variables . . . 327
27.5 Compiling and Running . . . 329
27.6 Summary . . . 329
References . . . 331
Homework (Code) . . . 332

28 Locks 333
28.1 Locks: The Basic Idea . . . 333
28.2 Pthread Locks . . . 334
28.3 Building A Lock . . . 335
28.4 Evaluating Locks . . . 335
28.5 Controlling Interrupts . . . 336
28.6 A Failed Attempt: Just Using Loads/Stores . . . 337
28.7 Building Working Spin Locks with Test-And-Set . . . 338
28.8 Evaluating Spin Locks . . . 341
28.9 Compare-And-Swap . . . 342
28.10 Load-Linked and Store-Conditional . . . 343
28.11 Fetch-And-Add . . . 344
28.12 Too Much Spinning: What Now? . . . 345
28.13 A Simple Approach: Just Yield, Baby . . . 346
28.14 Using Queues: Sleeping Instead Of Spinning . . . 347
28.15 Different OS, Different Support . . . 350
28.16 Two-Phase Locks . . . 352
28.17 Summary . . . 352
References . . . 353

Homework (Simulation) 354

29 Lock-based Concurrent Data Structures 355
29.1 Concurrent Counters 355
29.2 Concurrent Linked Lists 361
29.3 Concurrent Queues 364
29.4 Concurrent Hash Table 366
29.5 Summary 366
References 369
Homework (Code) 370

30 Condition Variables 371
30.1 Definition and Routines 372
30.2 The Producer/Consumer (Bounded Buffer) Problem 376
30.3 Covering Conditions 384
30.4 Summary 386
References 387
Homework (Code) 388

31 Semaphores 391
31.1 Semaphores: A Definition 391
31.2 Binary Semaphores (Locks) 393
31.3 Semaphores For Ordering 394
31.4 The Producer/Consumer (Bounded Buffer) Problem 396
31.5 Reader-Writer Locks 401
31.6 The Dining Philosophers 403
31.7 Thread Throttling 406
31.8 How To Implement Semaphores 406
31.9 Summary 407
References 409
Homework (Code) 410

32 Common Concurrency Problems 411
32.1 What Types Of Bugs Exist? 411
32.2 Non-Deadlock Bugs 412
32.3 Deadlock Bugs 415
32.4 Summary 424
References 425
Homework (Code) 426

33 Event-based Concurrency (Advanced) 427
33.1 The Basic Idea: An Event Loop 427
33.2 An Important API: `select()` (or `poll()`) 428
33.3 Using `select()` 429
33.4 Why Simpler? No Locks Needed 431
33.5 A Problem: Blocking System Calls 431
33.6 A Solution: Asynchronous I/O 432

33.7 Another Problem: State Management 433
33.8 What Is Still Difficult With Events 435
33.9 Summary 436
References 437
Homework (Code) 438

34 Summary Dialogue on Concurrency 439

III Persistence 441

35 A Dialogue on Persistence 443

36 I/O Devices 445
36.1 System Architecture 445
36.2 A Canonical Device 447
36.3 The Canonical Protocol 448
36.4 Lowering CPU Overhead With Interrupts 449
36.5 More Efficient Data Movement With DMA 450
36.6 Methods Of Device Interaction 451
36.7 Fitting Into The OS: The Device Driver 452
36.8 Case Study: A Simple IDE Disk Driver 453
36.9 Historical Notes 455
36.10 Summary 457
References 458

37 Hard Disk Drives 459
37.1 The Interface 459
37.2 Basic Geometry 460
37.3 A Simple Disk Drive 461
37.4 I/O Time: Doing The Math 464
37.5 Disk Scheduling 468
37.6 Summary 472
References 473
Homework (Simulation) 474

38 Redundant Arrays of Inexpensive Disks (RAIDs) 475
38.1 Interface And RAID Internals 476
38.2 Fault Model 477
38.3 How To Evaluate A RAID 477
38.4 RAID Level 0: Striping 478
38.5 RAID Level 1: Mirroring 481
38.6 RAID Level 4: Saving Space With Parity 484
38.7 RAID Level 5: Rotating Parity 488
38.8 RAID Comparison: A Summary 489
38.9 Other Interesting RAID Issues 490
38.10 Summary 490

References 491
Homework (Simulation) 492

39 Interlude: Files and Directories 493
39.1 Files And Directories 493
39.2 The File System Interface 495
39.3 Creating Files 495
39.4 Reading And Writing Files 497
39.5 Reading And Writing, But Not Sequentially 499
39.6 Shared File Table Entries: `fork()` And `dup()` 501
39.7 Writing Immediately With `fsync()` 504
39.8 Renaming Files 504
39.9 Getting Information About Files 506
39.10 Removing Files 507
39.11 Making Directories 508
39.12 Reading Directories 509
39.13 Deleting Directories 510
39.14 Hard Links 510
39.15 Symbolic Links 512
39.16 Permission Bits And Access Control Lists 514
39.17 Making And Mounting A File System 516
39.18 Summary 518
References 520
Homework (Code) 521

40 File System Implementation 523
40.1 The Way To Think 523
40.2 Overall Organization 524
40.3 File Organization: The Inode 526
40.4 Directory Organization 530
40.5 Free Space Management 532
40.6 Access Paths: Reading and Writing 532
40.7 Caching and Buffering 536
40.8 Summary 538
References 539
Homework (Simulation) 540

41 Locality and The Fast File System 541
41.1 The Problem: Poor Performance 541
41.2 FFS: Disk Awareness Is The Solution 543
41.3 Organizing Structure: The Cylinder Group 543
41.4 Policies: How To Allocate Files and Directories 545
41.5 Measuring File Locality 547
41.6 The Large-File Exception 548
41.7 A Few Other Things About FFS 550
41.8 Summary 552
References 553

Homework (Simulation) 554

42 Crash Consistency: FSCK and Journaling 555
42.1 A Detailed Example 556
42.2 Solution #1: The File System Checker 559
42.3 Solution #2: Journaling (or Write-Ahead Logging) 561
42.4 Solution #3: Other Approaches 571
42.5 Summary 572
References 573
Homework (Simulation) 575

43 Log-structured File Systems 577
43.1 Writing To Disk Sequentially 578
43.2 Writing Sequentially And Effectively 579
43.3 How Much To Buffer? 580
43.4 Problem: Finding Inodes 581
43.5 Solution Through Indirection: The Inode Map 581
43.6 Completing The Solution: The Checkpoint Region 583
43.7 Reading A File From Disk: A Recap 583
43.8 What About Directories? 584
43.9 A New Problem: Garbage Collection 585
43.10 Determining Block Liveness 586
43.11 A Policy Question: Which Blocks To Clean, And When? . . 587
43.12 Crash Recovery And The Log 588
43.13 Summary 588
References 590
Homework (Simulation) 591

44 Flash-based SSDs 593
44.1 Storing a Single Bit 593
44.2 From Bits to Banks/Planes 594
44.3 Basic Flash Operations 595
44.4 Flash Performance And Reliability 597
44.5 From Raw Flash to Flash-Based SSDs 598
44.6 FTL Organization: A Bad Approach 599
44.7 A Log-Structured FTL 600
44.8 Garbage Collection 602
44.9 Mapping Table Size 604
44.10 Wear Leveling 609
44.11 SSD Performance And Cost 609
44.12 Summary 611
References 613
Homework (Simulation) 615

45 Data Integrity and Protection 617
45.1 Disk Failure Modes 617
45.2 Handling Latent Sector Errors 619

45.3 Detecting Corruption: The Checksum 620
45.4 Using Checksums 623
45.5 A New Problem: Misdirected Writes 624
45.6 One Last Problem: Lost Writes 625
45.7 Scrubbing 625
45.8 Overheads Of Checksumming 626
45.9 Summary 627
References 628
Homework (Simulation) 629
Homework (Code) 630

46 Summary Dialogue on Persistence 631

47 A Dialogue on Distribution 633

48 Distributed Systems 635
48.1 Communication Basics 636
48.2 Unreliable Communication Layers 637
48.3 Reliable Communication Layers 639
48.4 Communication Abstractions 642
48.5 Remote Procedure Call (RPC) 643
48.6 Summary 648
References 649
Homework (Code) 650

49 Sun's Network File System (NFS) 653
49.1 A Basic Distributed File System 654
49.2 On To NFS 655
49.3 Focus: Simple And Fast Server Crash Recovery 655
49.4 Key To Fast Crash Recovery: Statelessness 656
49.5 The NFSv2 Protocol 657
49.6 From Protocol To Distributed File System 659
49.7 Handling Server Failure With Idempotent Operations . . . 661
49.8 Improving Performance: Client-side Caching 663
49.9 The Cache Consistency Problem 663
49.10 Assessing NFS Cache Consistency 665
49.11 Implications On Server-Side Write Buffering 665
49.12 Summary 667
References 669
Homework (Measurement) 670

50 The Andrew File System (AFS) 671
50.1 AFS Version 1 671
50.2 Problems with Version 1 673
50.3 Improving the Protocol 674
50.4 AFS Version 2 674
50.5 Cache Consistency 676

50.6 Crash Recovery 678
50.7 Scale And Performance Of AFSv2 679
50.8 AFS: Other Improvements 681
50.9 Summary 682
References 683
Homework (Simulation) 684

51 Summary Dialogue on Distribution 685

General Index 687

Asides 699

Tips 703

Cruces 707