## Contents

**Preface**

**Introduction**

1. What Is An Operating System?
   - Resource Sharing: Operating System as Referee 6
   - Masking Limitations: Operating System as Illusionist 9
   - Providing Common Services: Operating System as Glue 12
   - Operating System Design Patterns 13

1.2 Operating System Evaluation
   - Reliability and Availability 19
   - Security 20
   - Portability 21
   - Performance 23
   - Adoption 24
   - Design Tradeoffs 25

1.3 Operating Systems: Past, Present, and Future
   - Impact of Technology Trends 26
   - Early Operating Systems 27
   - Multi-User Operating Systems 28
   - Time-Sharing Operating Systems 29
   - Modern Operating Systems 30
   - Future Operating Systems 32

**Exercises** 33

## Part I  Kernels and Processes

### The Kernel Abstraction

2.1 The Process Abstraction 43

2.2 Dual-Mode Operation
   - Privileged Instructions 47
   - Memory Protection 49
   - Timer Interrupts 56

2.3 Types of Mode Transfer
   - User to Kernel Mode 57
   - Kernel to User Mode 61

2.4 Implementing Safe Mode Transfer
   - Interrupt Vector Table 63
   - Interrupt Stack 64
   - Two Stacks per Process 65
   - Interrupt Masking 67
   - Hardware Support for Saving and Restoring Registers 68

2.5 Putting It All Together: x86 Mode Transfer 69
### 2.6 Implementing Secure System Calls

### 2.7 Starting a New Process

### 2.8 Implementing Upcalls

### 2.9 Case Study: Booting an Operating System Kernel

### 2.10 Case Study: Virtual Machines

### 2.11 Summary and Future Directions

### Exercises

### 3 The Programming Interface

#### 3.1 Process Management
- Windows Process Management
- UNIX Process Management

#### 3.2 Input/Output

#### 3.3 Case Study: Implementing a Shell

#### 3.4 Case Study: Interprocess Communication
- Producer-Consumer Communication
- Client-Server Communication

#### 3.5 Operating System Structure
- Monolithic Kernels
- Microkernel

#### 3.6 Summary and Future Directions

### Exercises

### Part II Concurrency

### 4 Concurrency and Threads

#### 4.1 Thread Use Cases
- Four Reasons to Use Threads

#### 4.2 Thread Abstraction
- Running, Suspending, and Resuming Threads
- Why “Unpredictable Speed”?

#### 4.3 Simple Thread API
- A Multi-Threaded Hello World
- Fork-Join Parallelism

#### 4.4 Thread Data Structures and Life Cycle
- Per-Thread State and Thread Control Block (TCB)
- Shared State

#### 4.5 Thread Life Cycle

#### 4.6 Implementing Kernel Threads
- Creating a Thread
- Deleting a Thread
- Thread Context Switch

#### 4.7 Combining Kernel Threads and Single-Threaded User Processes

#### 4.8 Implementing Multi-Threaded Processes
- Implementing Multi-Threaded Processes Using Kernel Threads
- Implementing User-Level Threads Without Kernel Support
Implementing User-Level Threads With Kernel Support 168

4.9 Alternative Abstractions 170
   Asynchronous I/O and Event-Driven Programming 171
   Data Parallel Programming 175

4.10 Summary and Future Directions 176
   Historical Notes 177
Exercises 178

5

Synchronizing Access to Shared Objects 183

5.1 Challenges 187
   Race Conditions 187
   Atomic Operations 189
   Too Much Milk 190
   Discussion 193
   A Better Solution 194

5.2 Structuring Shared Objects 195
   Implementing Shared Objects 196
   Scope and Roadmap 198

5.3 Locks: Mutual Exclusion 198
   Locks: API and Properties 199
   Case Study: Thread-Safe Bounded Queue 201

5.4 Condition Variables: Waiting for a Change 205
   Condition Variable Definition 206
   Thread Life Cycle Revisited 211
   Case Study: Blocking Bounded Queue 213

5.5 Designing and Implementing Shared Objects 214
   High Level Methodology 216
   Implementation Best Practices 220
   Three Pitfalls 224

5.6 Three Case Studies 227
   Readers/ Writers Lock 227
   Synchronization Barriers 231
   FIFO Blocking Bounded Queue 236

5.7 Implementing Synchronization Objects 237
   Implementing Uniprocessor Locks by Disabling Interrupts 238
   Implementing Uniprocessor Queuing Locks 239
   Implementing Multiprocessor Spinlocks 240
   Implementing Multiprocessor Queuing Locks 240
   Case Study: Linux 2.6 Kernel Mutex Lock 244
   Implementing Condition Variables 246
   Implementing Application-level Synchronization 246

5.8 Semaphores Considered Harmful 248

5.9 Summary and Future Directions 253
   Historical Notes 254
Exercises 254

6

Multi-Object Synchronization 261

6.1 Multiprocessor Lock Performance 262

6.2 Lock Design Patterns 266
Operating Systems: Principles and Practice

6.3 Lock Contention 272
  MCS Locks 272
  Read-Copy-Update (RCU) 274

6.4 Multi-Object Atomicity 282
  Careful Class Design 283
  Acquire-All/Release-All 283
  Two-Phase Locking 284

6.5 Deadlock 285
  Deadlock vs. Starvation 288
  Necessary Conditions for Deadlock 289
  Preventing Deadlock 291
  The Banker’s Algorithm for Avoiding Deadlock 294
  Detecting and Recovering From Deadlocks 299

6.6 Non-Blocking Synchronization 305

6.7 Summary and Future Directions 307

Exercises 308

7 Scheduling 313

7.1 Uniprocessor Scheduling 316
  First-In-First-Out (FIFO) 317
  Shortest Job First (SJF) 318
  Round Robin 320
  Max-Min Fairness 323
  Case Study: Multi-Level Feedback 326
  Summary 328

7.2 Multiprocessor Scheduling 329
  Scheduling Sequential Applications on Multiprocessors 329
  Scheduling Parallel Applications 331

7.3 Energy-Aware Scheduling 337

7.4 Real-Time Scheduling 340

7.5 Queueing Theory 343
  Definitions 343
  Little’s Law 345
  Response Time Versus Utilization 348
  “What if?” Questions 355
  Lessons 358

7.6 Overload Management 358

7.7 Case Study: Servers in a Data Center 361

7.8 Summary and Future Directions 362

Exercises 363

Part III Memory Management 371

8 Address Translation 371
# Operating Systems: Principles and Practice

8.1 Address Translation Concept 374
8.2 Towards Flexible Address Translation 376
  Segmented Memory 376
  Paged Memory 382
  Multi-Level Translation 385
  Portability 390
8.3 Towards Efficient Address Translation 392
  Translation Lookaside Buffers 393
  Superpages 396
  TLB Consistency 398
  Virtually Addressed Caches 402
  Physically Addressed Caches 403
8.4 Software Protection 405
  Single Language Operating Systems 406
  Language-Independent Software Fault Isolation 410
  Sandboxes Via Intermediate Code 413
8.5 Summary and Future Directions 414
Exercises 415

9 Caching and Virtual Memory 421
9.1 Cache Concept 425
9.2 Memory Hierarchy 427
9.3 When Caches Work and When They Do Not 429
  Working Set Model 430
  Zipf Model 433
9.4 Memory Cache Lookup 434
9.5 Replacement Policies 439
  Random 439
  First-In-First-Out (FIFO) 440
  Optimal Cache Replacement (MIN) 441
  Least Recently Used (LRU) 442
  Least Frequently Used (LFU) 443
  Belady's Anomaly 444
9.6 Case Study: Memory-Mapped Files 445
  Advantages 446
  Implementation 447
  Approximating LRU 451
9.7 Case Study: Virtual Memory 454
  Self-Paging 454
  Swapping 456
9.8 Summary and Future Directions 457
Exercises 458

10 Advanced Memory Management 465
10.1 Zero-Copy I/O 467
10.2 Virtual Machines 470
  Virtual Machine Page Tables 470
  Transparent Memory Compression 472
10.3 Fault Tolerance 475
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkpoint and Restart</td>
<td>476</td>
</tr>
<tr>
<td>Recoverable Virtual Memory</td>
<td>478</td>
</tr>
<tr>
<td>Deterministic Debugging</td>
<td>479</td>
</tr>
<tr>
<td>10.4 Security</td>
<td>481</td>
</tr>
<tr>
<td>10.5 User-Level Memory Management</td>
<td>482</td>
</tr>
<tr>
<td>10.6 Summary and Future Directions</td>
<td>485</td>
</tr>
<tr>
<td>Exercises</td>
<td>485</td>
</tr>
</tbody>
</table>

## Part IV Persistent Storage

### 11 File Systems: Introduction and Overview

11.1 The File System Abstraction 495
11.2 API 502
11.3 Software Layers 505
   - API and Performance 506
   - Device Drivers: Common Abstractions 508
   - Device Access 508
   - Putting It All Together: A Simple Disk Request 511
11.4 Summary and Future Directions 511
Exercises 513

### 12 Storage Devices

12.1 Magnetic Disk 517
   - Disk Access and Performance 520
   - Case Study: Toshiba MK3254GSY 523
   - Disk Scheduling 527
12.2 Flash Storage 530
12.3 Summary and Future Directions 536
Exercises 540

### 13 Files and Directories

13.1 Implementation Overview 546
13.2 Directories: Naming Data 547
13.3 Files: Finding Data 553
   - FAT: Linked List 555
   - FFS: Fixed Tree 558
   - NTFS: Flexible Tree With Extents 568
   - Copy-On-Write File Systems 573
13.4 Putting It All Together: File and Directory Access 581
13.5 Summary and Future Directions 583
Exercises 584

### 14 Reliable Storage

14.1 Transactions: Atomic Updates 592
   - Ad Hoc Approaches 593
   - The Transaction Abstraction 595
   - Implementing Transactions 597
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions and File Systems</td>
<td>609</td>
</tr>
<tr>
<td>14.2  Error Detection and Correction</td>
<td>613</td>
</tr>
<tr>
<td>Storage Device Failures and Mitigation</td>
<td>613</td>
</tr>
<tr>
<td>RAID: Multi-Disk Redundancy for Error Correction</td>
<td>619</td>
</tr>
<tr>
<td>Software Integrity Checks</td>
<td>631</td>
</tr>
<tr>
<td>14.3  Summary and Future Directions</td>
<td>633</td>
</tr>
<tr>
<td>Exercises</td>
<td>635</td>
</tr>
<tr>
<td>References</td>
<td>641</td>
</tr>
<tr>
<td>Glossary</td>
<td>651</td>
</tr>
<tr>
<td>Index</td>
<td>662</td>
</tr>
<tr>
<td>About the Authors</td>
<td>669</td>
</tr>
</tbody>
</table>
An Operating System (OS) is a set of computer programs that manage the hardware and software resources of a computer. Popular Operating Systems include Microsoft Windows, Mac OS X and Linux. An operating system performs basic system tasks such as controlling input and output devices, facilitating networking, managing file systems, managing applications, allocating memory and prioritizing system requests. Zorin OS 15.2, released on March 8, adds an impressive selection of upgrades and improvements to an already well-oiled Linux operating system. Since its debut in July 2009 Zorin OS cofounder Artyom Zorin has hawked his distribution as an ideal Microsoft Windows replacement. That description is a strong selling point for this easier-to-use computing platform.