

Principles of Operating Systems: Design & Applications

Brian L. Stuart
FedEx Labs
University of Memphis

Australia · Canada · Mexico · Singapore · Spain · United Kingdom · United States

Principles of Operating Systems: Design & Applications

Brian L. Stuart

Senior Product Manager:
Alyssa Pratt

Marketing Manager:
Bryant Chrzan

Cover Design:
Yvo Riezebos Designs

Acquisitions Editor:
Amy Jollymore

Editorial Assistant:
Patrick Frank

Art Director:
Beth Paquin

Development Editor:
Jim Markham

Manufacturing Coordinator:
Julio Esperas

Compositor:
Brian L. Stuart

Content Project Manager:
Matt Hutchinson

Printer:
RR Donnelley, Crawfordsville

COPYRIGHT ©2009 Course Technology, a division of Thomson Learning, Inc.
Thomson Learning™ is a trademark used herein under license.

Printed in the United States of America

For more information, contact Course Technology, 25 Thomson Place,
Boston, Massachusetts, 02210.

Or find us on the World Wide Web at: www.course.com

ALL RIGHTS RESERVED. No part of this work covered by the copyright herein may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, Web distribution, or information storage and retrieval systems—without the written permission of the publisher.

For permission to use material from this text or product, submit a request online at <http://www.thomsonrights.com>.

Any additional questions about permission can be submitted by email to thomsonrights@thomson.com.

Disclaimer

Course Technology reserves the right to revise this publication and make changes from time to time in its content without notice.

ISBN 1-4188-3769-5

for my beloved wife and daughter

Brief Contents

Preface	xxi
1 Introduction to Operating Systems	1
2 Some Example Operating Systems	21
3 Inferno Structure and Initialization	35
4 Linux Structure and Initialization	55
5 Principles of Process Management	81
6 Some Examples of Process Management	121
7 Process Management in Inferno	141
8 Process Management in Linux	165
9 Principles of Memory Management	195
10 Some Examples of Memory Management	233
11 Memory Management in Inferno	253
12 Memory Management in Linux	281
13 Principles of I/O Device Management	307
14 Some Examples of I/O Device Management	333
15 I/O Devices in Inferno	345
16 I/O Devices in Linux	371
17 Principles of File Systems	399
18 Some Examples of File Systems	425
19 File Systems in Inferno	437

20 File Systems in Linux	479
21 Principles of Operating System Security	511
22 Principles of Distributed Systems	537
A Compiling Hosted Inferno	555
B Compiling Native Inferno	561
Suggested Readings	565
Index	569

Contents

Preface	xxi
1 Introduction to Operating Systems	1
1.1 What Is an Operating System?	1
1.1.1 Resource Manager	2
1.1.2 Service Provider	3
1.1.3 Virtual Machine	3
1.2 Areas of Operating System Responsibility	3
1.2.1 Processes	4
1.2.2 Memory	4
1.2.3 I/O Devices	5
1.2.4 File Systems	6
1.2.5 Security	6
1.2.6 Networking	7
1.2.7 User Interfaces	8
1.3 History of Operating Systems	8
1.3.1 Bare Metal	9
1.3.2 Batch Operating Systems	9
1.3.3 Time-Sharing Operating Systems	10
1.3.4 Distributed Operating Systems	11
1.4 Techniques of Organizing Operating Systems	12
1.4.1 Monolithic Designs	12
1.4.2 Layered Designs	12
1.4.3 Microkernel Designs	12
1.4.4 Virtual Machine Designs	14
1.5 Bootstrapping	15
1.6 System Calls	17
1.6.1 Example System Calls	17
1.6.2 System Call Mechanism	18
1.7 Summary	18
1.8 Exercises	19
2 Some Example Operating Systems	21
2.1 CTSS	21
2.1.1 Organization	22
2.1.2 Booting	22

2.2	Multics	22
2.2.1	Organization	23
2.2.2	System Calls	24
2.3	RT-11	24
2.3.1	Organization	24
2.4	Sixth Edition UNIX	26
2.4.1	Organization	27
2.4.2	System Calls	27
2.5	VMS	27
2.5.1	Organization	28
2.5.2	Booting	28
2.5.3	System Calls	29
2.6	4.3BSD	29
2.6.1	Organization	30
2.6.2	System Calls	30
2.7	Windows NT	30
2.7.1	Organization	31
2.8	TinyOS	31
2.8.1	Organization	32
2.9	Xen	32
2.9.1	Organization	32
2.10	Summary	33
2.11	Exercises	33
3	Inferno Structure and Initialization	35
3.1	Origins of Inferno	35
3.2	Fundamental Concepts	36
3.3	Organization	39
3.3.1	Basic Architecture	39
3.3.2	Source Code Organization	40
3.4	Initialization	42
3.4.1	Starting Inferno	43
3.4.2	Host OS Specific Initialization	45
3.4.3	Host OS Independent Initialization	47
3.4.4	Starting Time-Sharing	50
3.5	System Calls	52
3.6	Summary	52
3.7	Exercises	53
4	Linux Structure and Initialization	55
4.1	The Origins of Linux	55
4.2	Organization	57
4.2.1	Basic Architecture	57
4.2.2	Modules	58

4.2.3	Source Code Organization	59
4.3	Initialization	61
4.3.1	Bootstrapping	62
4.3.2	Processor-Specific Initialization	65
4.3.3	Processor-Independent Initialization	68
4.3.4	Starting Time-Sharing	73
4.3.5	Initiating Administrative Initialization	74
4.4	System Calls	76
4.4.1	Application-Side System Call Handling	76
4.4.2	Kernel-Side System Call Handling	77
4.5	Summary	78
4.6	Exercises	78
5	Principles of Process Management	81
5.1	The Process Concept	81
5.2	Realizing Processes	82
5.2.1	Process Operations	82
5.2.2	Process State	84
5.2.3	The Process Table	85
5.3	Threads	86
5.4	Scheduling	86
5.4.1	First-Come, First-Served	87
5.4.2	Shortest Job First	87
5.4.3	Round-Robin	90
5.4.4	Priority Scheduling	90
5.4.5	Adjusting Scheduling Parameters	94
5.4.6	Two-Level Scheduling	94
5.4.7	Real-Time Scheduling	94
5.4.8	Scheduling in Embedded Systems	97
5.5	Context Switching	97
5.6	Process Creation and Termination	100
5.7	Critical Sections	101
5.7.1	Interrupt Control	102
5.7.2	Atomic Instructions	103
5.7.3	Peterson's Algorithm	104
5.7.4	Semaphores	105
5.7.5	Monitors	106
5.7.6	Message Passing	107
5.7.7	Examples	107
5.8	Deadlock	111
5.8.1	Necessary and Sufficient Conditions	111
5.8.2	Dealing with Deadlock	112
5.9	Summary	118
5.10	Exercises	118

6	Some Examples of Process Management	121
6.1	CTSS	121
6.1.1	Process State	121
6.1.2	System Calls	122
6.1.3	Scheduling	123
6.2	Multics	124
6.2.1	System Calls	124
6.2.2	Process State	124
6.2.3	Scheduling	125
6.3	RT-11	126
6.3.1	System Calls	126
6.3.2	Process State	126
6.3.3	Process Table	126
6.3.4	Scheduling	127
6.4	Sixth Edition UNIX	127
6.4.1	System Calls	127
6.4.2	Process State	128
6.4.3	Process Table	128
6.4.4	Scheduling	128
6.5	4.3BSD	129
6.5.1	System Calls	129
6.5.2	Process State and Process Table	130
6.5.3	Scheduling	130
6.6	VMS	131
6.6.1	System Calls	131
6.6.2	Thread State	132
6.6.3	Scheduling	132
6.7	Windows NT	133
6.7.1	System Calls	133
6.7.2	Thread State	134
6.7.3	Process and Thread Tables	134
6.7.4	Scheduling	135
6.8	TinyOS	136
6.9	Xen	137
6.10	Summary	138
6.11	Exercises	138
7	Process Management in Inferno	141
7.1	Processes in Inferno	141
7.2	Process State	142
7.2.1	Kernel Processes	142
7.2.2	User Processes	143
7.3	Process Data Structures	145
7.3.1	Kernel Process Table	145

7.3.2	Kernel Process Table Entry	147
7.3.3	User Process Table	148
7.3.4	User Process Table Entry	148
7.4	Process Creation	151
7.4.1	Interpreting the Process Creation Instruction	151
7.4.2	Implementing Process Creation	152
7.5	Process Destruction	155
7.6	Process Scheduling	157
7.6.1	Adding to the Ready List	157
7.6.2	Removing from the Ready List	158
7.6.3	Time-Sharing	159
7.6.4	Running a Time Slice	161
7.7	Summary	163
7.8	Exercises	163
8	Process Management in Linux	165
8.1	Process and Threads	165
8.1.1	Kernel Threads in Linux	165
8.1.2	Process Relationships	166
8.2	System Calls	166
8.3	Process State	168
8.4	Process Table	170
8.5	Process Creation	173
8.5.1	Handling the System Call	173
8.5.2	Creating the Process	176
8.5.3	Architecture-Specific Steps	178
8.6	Process Scheduling	181
8.6.1	Priorities	182
8.6.2	Queue Structure	182
8.6.3	Clock Ticks	184
8.6.4	Scheduler	187
8.7	Summary	193
8.8	Exercises	193
9	Principles of Memory Management	195
9.1	The Memory Hierarchy	195
9.2	Address Translation	197
9.2.1	Base/Limit Registers	198
9.2.2	Segmentation	198
9.2.3	Paging	199
9.3	Memory-Related Services	202
9.4	Memory Layouts	203
9.5	Memory Allocation Techniques	206
9.5.1	Free Space Management	206

9.5.2	Fragmentation	208
9.5.3	Partitioning	208
9.5.4	Selection Policies	209
9.5.5	Buddy System Management	211
9.6	Overallocation Techniques	213
9.6.1	Swapping	214
9.6.2	Segment Swapping	214
9.6.3	Paging	215
9.6.4	Paged Segments	224
9.6.5	Memory-Mapped Files	225
9.6.6	Copy on Write	226
9.6.7	Performance Issues	226
9.7	Memory Management in Embedded Systems	229
9.8	Summary	229
9.9	Exercises	230
10	Some Examples of Memory Management	233
10.1	CTSS	233
10.2	Multics	234
10.2.1	Memory-Related System Calls	234
10.2.2	Memory Layouts	235
10.2.3	Segment and Page Management	235
10.3	RT-11	235
10.3.1	Memory-Related System Calls	236
10.3.2	Memory Layouts	236
10.3.3	USR and KMON Swapping	237
10.4	Sixth Edition UNIX	238
10.4.1	Memory-Related System Calls	238
10.4.2	Memory Layouts	238
10.4.3	Free Space Management	240
10.4.4	Allocation	240
10.4.5	Swapping	240
10.5	4.3BSD	241
10.5.1	Memory-Related System Calls	241
10.5.2	Memory Layouts	241
10.5.3	Free Space Management	243
10.5.4	Swapping and Page Replacement	243
10.6	VMS	245
10.6.1	Page Tables	245
10.6.2	Memory Layouts	245
10.6.3	Free Space Management	245
10.6.4	Swapping and Page Replacement	246
10.6.5	Memory-Related System Calls	247
10.7	Windows NT	247

10.7.1	System Calls	248
10.7.2	Memory Layouts	248
10.7.3	Page Management	248
10.8	TinyOS	250
10.9	Xen	250
10.9.1	Hypercalls	250
10.9.2	Memory Layouts	250
10.9.3	Page Management	251
10.10	Summary	251
10.11	Exercises	251
11	Memory Management in Inferno	253
11.1	Overview	253
11.2	Memory Layouts	255
11.3	Memory Management Data Structures	255
11.3.1	Memory Pools	256
11.3.2	Memory Blocks	258
11.4	Memory Management Implementation	261
11.4.1	Allocating Memory	261
11.4.2	Removing a Free Block from the Tree	267
11.4.3	Freeing Memory	270
11.4.4	Inserting a Free Block into the Tree	271
11.5	Garbage Collection	273
11.5.1	Heap Structure	273
11.5.2	Reference Counts	273
11.5.3	Very Concurrent Garbage Collector	273
11.5.4	Implementing VCGC	275
11.6	Summary	279
11.7	Exercises	279
12	Memory Management in Linux	281
12.1	Memory Layouts	281
12.2	System Calls	284
12.3	Allocation Mechanisms	285
12.3.1	Zoned Page Allocation	285
12.3.2	Slab Allocator	285
12.3.3	Kernel Memory Allocation	286
12.4	Page Management	286
12.4.1	Page Tables	286
12.4.2	Page Replacement	287
12.5	Memory Management Data Structures	288
12.5.1	Representing Process Allocation	288
12.5.2	Representing Virtual Memory Areas	290
12.6	Memory Management Implementation	292

12.6.1	Handling the Allocation System Call	292
12.6.2	Adding a Region	295
12.6.3	Processing Page Faults	298
12.6.4	Resolving a Fault for a Missing Page	300
12.6.5	Handling New Page Frames	302
12.7	Summary	305
12.8	Exercises	305
13	Principles of I/O Device Management	307
13.1	Elements of the I/O Subsystem	307
13.2	I/O Device Hardware Characteristics	309
13.2.1	Disk Drives	309
13.2.2	Serial Communications	312
13.2.3	Controller Interface Techniques	314
13.3	Types of I/O Devices	317
13.3.1	Communication vs. Storage Devices	317
13.3.2	Stream vs. Block Devices	318
13.4	Objectives of I/O Subsystem Design	319
13.5	I/O Device Services	319
13.6	Device Driver Structure	320
13.7	Device Management Techniques	322
13.7.1	Buffering	322
13.7.2	Interleaving	323
13.7.3	Elevator Algorithm	324
13.7.4	RAID	326
13.7.5	Water Marks	328
13.7.6	Human Input Processing	330
13.7.7	Pseudo-Devices	331
13.8	Summary	331
13.9	Exercises	331
14	Some Examples of I/O Device Management	333
14.1	CTSS	333
14.2	Multics	334
14.3	RT-11	335
14.4	Sixth Edition UNIX	338
14.5	4.3BSD	339
14.6	VMS	340
14.7	Windows NT	341
14.8	TinyOS	342
14.9	Xen	342
14.10	Summary	343
14.11	Exercises	343

15 I/O Devices in Inferno	345
15.1 Device Driver Structure	345
15.2 Parallel Port Support	347
15.2.1 Servicing a Write Request	347
15.2.2 Writing a Single Byte	349
15.3 Keyboard Support	350
15.3.1 Initializing the Keyboard Controller	352
15.3.2 Handling a Keyboard Interrupt	353
15.4 IDE Disk Support	357
15.4.1 Processing an I/O Request	359
15.4.2 Initiating an IDE Controller Operation	362
15.4.3 Handling an IDE Controller Interrupt	366
15.5 Summary	369
15.6 Exercises	369
16 I/O Devices in Linux	371
16.1 Block Request Support	371
16.2 Two-Half Interrupt Handler Structure	372
16.3 Parallel Port Driver	374
16.3.1 Handling the System Call	374
16.3.2 Selecting the Proper Low-Level Write	378
16.3.3 Writing Bytes from the Buffer	380
16.3.4 Configuring the Controller	383
16.4 Floppy Disk Driver	385
16.4.1 Handling the Request	385
16.4.2 Scheduling the Floppy Operation	386
16.4.3 Performing a Floppy Operation	387
16.4.4 Starting the Command	390
16.4.5 Preparing for the Data Transfer	390
16.4.6 Programming the Controller	392
16.4.7 Handling a Floppy Interrupt	394
16.4.8 Finishing the Floppy Operation	395
16.5 Summary	397
16.6 Exercises	397
17 Principles of File Systems	399
17.1 File System Services	399
17.1.1 Shared and Exclusive Access	400
17.1.2 Access Patterns	401
17.1.3 File Structure	401
17.1.4 Metadata	403
17.1.5 Memory-Mapped Files	403
17.2 General File System Design	404
17.2.1 Form of the File System	404

17.2.2	Major Data Structures	405
17.3	Name Spaces	406
17.3.1	Drive Specifiers	407
17.3.2	Account Specifiers	408
17.3.3	Hierarchical Naming	409
17.3.4	File Extensions	410
17.3.5	File Versions	411
17.3.6	Special Files and Directories	411
17.3.7	Relative and Absolute Names	412
17.4	Managing Storage Space	412
17.4.1	File System Metadata	412
17.4.2	Data Units	413
17.4.3	Free Space Management	413
17.4.4	Regular Files	415
17.4.5	Sparse Files	418
17.4.6	Forks	418
17.4.7	Directories	419
17.4.8	Aliases	419
17.5	Consistency Checking	420
17.6	Journaling and Log-Structured File Systems	421
17.7	Block Caching	422
17.8	Summary	423
17.9	Exercises	424
18	Some Examples of File Systems	425
18.1	CTSS	425
18.1.1	First CTSS File System	425
18.1.2	Second CTSS File System	426
18.2	Multics	427
18.3	RT-11	428
18.4	Sixth Edition UNIX	429
18.5	4.3BSD	431
18.6	VMS	432
18.7	Windows NT	434
18.8	Summary	435
18.9	Exercises	435
19	File Systems in Inferno	437
19.1	The Role of File Servers	437
19.1.1	The Styx Protocol	438
19.1.2	Built-in Kernel File Servers	441
19.1.3	User-Space File Servers	441
19.2	The Root Device Server	442
19.2.1	Providing the Names Served	443

19.2.2	Walking the Root Server's Tree	444
19.2.3	Reading from the Root Server	445
19.3	Generic Styx Message Handler	445
19.3.1	Creating a Directory Entry	446
19.3.2	Generating Names	446
19.3.3	Walking a Directory Tree	447
19.4	Native Inferno File System	450
19.4.1	Initialization	451
19.4.2	Main Server Process	456
19.4.3	Processing a Styx Request	456
19.4.4	Walking a Directory Tree	458
19.4.5	Searching a Directory	460
19.4.6	Reading from a File	464
19.4.7	On-Disk Data Structures	466
19.4.8	Reading a Directory Entry	471
19.4.9	Reading a File Block	471
19.4.10	Locating a File Block	472
19.4.11	Processing Indirect Blocks	474
19.4.12	Fetching from the Buffer Cache	475
19.5	Summary	477
19.6	Exercises	478
20	File Systems in Linux	479
20.1	Virtual File System	479
20.1.1	Superblocks	480
20.1.2	I-Nodes	480
20.1.3	Directory Entries	481
20.1.4	Files	481
20.2	The EXT3 File System	481
20.3	EXT3 Disk Structure	482
20.3.1	EXT3 Superblock	482
20.3.2	EXT3 I-Node	485
20.3.3	EXT3 Directory Entries	487
20.4	EXT3 Name Lookup	488
20.4.1	Walking a Path	488
20.4.2	Generic Directory Lookup (Part 1)	494
20.4.3	Generic Directory Lookup (Part 2)	495
20.4.4	EXT3 Directory Lookup	496
20.4.5	EXT3 Directory Search	498
20.4.6	EXT3 Directory Block Search	501
20.5	File Writing	502
20.5.1	Linux Write System Call	502
20.5.2	Generic File Writing	503
20.5.3	EXT3 File Writing	505

20.6	Locating File Blocks in EXT3	505
20.6.1	Identifying the Indirect Blocks	506
20.6.2	Reading the Indirect Blocks	508
20.7	Summary	509
20.8	Exercises	509
21	Principles of Operating System Security	511
21.1	User Authentication	511
21.1.1	User Names and Passwords	512
21.1.2	Cryptographic Hashing Functions	512
21.1.3	Callbacks	513
21.1.4	Challenge/Response Authentication	513
21.1.5	One-Time Passwords	514
21.1.6	Biometric Authentication	514
21.2	Basic Resource Protection	515
21.2.1	Privileged Users	515
21.2.2	Access to CPU Features	516
21.2.3	Memory Access	517
21.2.4	Simple Protection Codes	517
21.2.5	Access Control Lists	519
21.2.6	Capabilities	520
21.3	Types of Threats	520
21.3.1	Man-in-the-Middle Attack	521
21.3.2	Trojan Horse	521
21.3.3	Trapdoor	521
21.3.4	Logic/Time Bomb	522
21.3.5	Virus	522
21.3.6	Worm	523
21.3.7	Covert Channel	523
21.3.8	Denial of Service	524
21.4	Orange Book Classification	524
21.4.1	Division D	525
21.4.2	Division C	525
21.4.3	Division B	526
21.4.4	Division A	527
21.5	Encryption	527
21.5.1	Symmetric Encryption	528
21.5.2	Public Key Cryptography	529
21.6	Protection Rings in Multics	531
21.7	Security in Inferno	533
21.8	Security in Linux	533
21.9	Summary	535
21.10	Exercises	535

22 Principles of Distributed Systems	537
22.1 Basic Concepts	537
22.1.1 Resource Sharing	538
22.1.2 Synchronous Operation	541
22.1.3 Consensus	541
22.1.4 Distributed Mutual Exclusion	541
22.1.5 Fault Tolerance	543
22.1.6 Self-Stabilization	543
22.2 Processor Sharing	544
22.2.1 Symmetric Multiprocessing	544
22.2.2 Clusters	545
22.2.3 Grids	546
22.3 Distributed Clocks	546
22.3.1 Logical Clocks	547
22.3.2 Physical Clocks	547
22.4 Election Algorithms	548
22.4.1 Bully Algorithm	548
22.4.2 Ring Algorithm	550
22.5 Summary	552
22.6 Exercises	553
A Compiling Hosted Inferno	555
A.1 Setting Up the Configuration	555
A.2 Compiler and Development Tools	557
A.3 The PATH Environment Variable	557
A.4 Other Environment Variables	558
A.5 Compiling the System	558
A.6 Running the New Version	559
A.7 Summary	559
B Compiling Native Inferno	561
B.1 Setting Up the Configuration	561
B.2 Building the Tool Chain	561
B.3 Building the Bootstrapping Code	562
B.4 Setting Up the Kernel Configuration	562
B.5 Creating the Loader Configuration	562
B.6 Building the Kernel Image	563
B.7 Making the Floppy Image	563
B.8 Running the New Kernel	563
B.9 Summary	564
Suggested Readings	565
Index	569

Preface

The seed for this book was planted over 20 years ago when I was in graduate school. During the summer of 1986, a group of students collected for an advanced operating systems seminar, under the leadership of Dr. David Cohen. We began with the intention of writing an operating system, but our focus soon shifted to writing a textbook on operating systems. As so often happens, little of our original intent was accomplished. However, along the way, we had many fruitful discussions about how to organize a book on operating systems and, by extension, how to approach teaching operating systems.

That, combined with a number of years teaching operating systems, led me to observe that there are several different approaches to teaching operating systems. It also led me to realize that no existing text really provided the instructor with the flexibility to draw on each of the different approaches as desired. The motivating objective for this book is to provide that flexibility.

Organization

Seven topics make up the body of this book. I begin with an introduction that highlights history, structure and organization, system calls, and bootstrapping. Following this introduction, I examine in some depth each of the major areas of operating system responsibility: processes, memory, I/O devices, and file systems. The final two topics are security and distributed systems.

The coverage of the first five topics is presented in sequences of four chapters each that examine each topic from a variety of perspectives. The first chapter in each sequence presents general principles associated with managing a resource. In these chapters, I introduce the relevant issues, and I present some standard techniques for addressing those issues. In some cases, the chapters devoted to general principles also include discussion of related issues. For example, the subjects of mutual exclusion and deadlock are discussed along with process management in Chapter 5 because of their relevance to interprocess communication. The second chapter in each sequence surveys a number of historic and current operating systems. The set of nine OS examples includes CTSS, Multics, RT-11, sixth edition UNIX, 4.3BSD, VMS, Windows NT, TinyOS, and Xen. My focus with these is to study in a high-level way how their developers translated available standard techniques into practice. In the third and fourth chapters of each part, I drill down further into implementational considerations. I discuss selected parts of the code for Inferno (in the third chapter) and Linux (in the fourth chapter).

However, I do not use this pattern in Chapters 21 and 22. These chapters discuss security and distributed systems, respectively. Because these topics are extensive enough for full books in their own right, I necessarily take a selective approach to them. These

chapters present only a few representative techniques. I also discuss a more restricted set of examples in Chapter 21, illustrating applied security techniques.

I describe how you build a hosted Inferno kernel image in Appendix A. Doing this is a part of implementing solutions to those assignments that ask for modifications to the Inferno kernel. A kernel image built in this way can be run as an application on an existing host OS. Appendix B shows you how to do the same for a native kernel. In particular, I provide the steps necessary to create a bootable floppy image for an x86 PC. This image can then be written to a floppy or used to create a bootable CD-ROM.

Intended Audience

The audience for this book consists of two groups. The first group is practitioners who want to learn about the internals of Linux or Inferno, as well as reinforce their understanding of the basic principles. People in this group will likely come to the book with substantial exposure and experience in their respective OS. They will also likely have some familiarity with some of the concepts and techniques of OS operation. There is a possibility that such readers might not have been exposed to some of the data structures discussed in this book. Any book on data structures will provide the necessary background. Likewise, books on computer organization can provide good background in computer hardware. For this group, the chapters on general principles will help fill in any gaps in their knowledge, and the respective chapters on Linux and Inferno will provide introductions to the internals of those operating systems.

The second group this book serves is instructors and students of operating systems classes. Both introductory and advanced OS courses can make use of this book. Typical prerequisites for OS courses include a data structures course and a computer organization course. Some sections of this book assume that kind of background. Other sections are connected to programming languages, their compilers, and their run-time environments. Although courses on programming languages and compilers are helpful background, they are not necessary to study this material.

Using the Book

A book is intended to be read straight through, from start to finish, in most cases. This book can be used in that way. However, most instructors will not use it in that way. Rather, the most effective way to use this book in the classroom is to take selected material from each of the major parts. The full range of material provides each instructor with the flexibility to select the material that best fits the style of the course and personal preferences.

An instructor of an operating system course will choose those sections that support the course's general pedagogical approach. For example, one approach might focus on concepts and techniques in the abstract. Coverage of the difficulties associated with implementing those techniques on real hardware is traded off for theoretical depth. Another approach might trade off time used to present general principles for time used to illustrate the application of principles through a survey of a number of real operating systems. The final common approach to introductory classes combines a study of general principles

with an in-depth examination of the implementation of an OS. In this last type of course, students are often required to make modifications to the operating system they study. Such in-depth experience with the internals of an existing OS is also often found in advanced operating systems classes.

Now, consider a set of recommended chapters and sections for each style of course. For general principles courses, the focus should be on a thorough coverage of Chapters 1, 5, 9, 13, 17, 21, and 22. Instructors might want to supplement this material with selected examples from Chapters 2, 6, 10, 14, and 18. Similarly, these chapters can be assigned as outside readings. One particular formulation of this type of course deserves special attention. The course designated CS220 in the Computing Curriculum 2001 specifies a number of specific topics that should be covered. The following sections provide good coverage of those recommendations: 1.1–1.4, 1.6, 5.1–5.8, 9.1–9.3, 9.5–9.6, 13.1–13.4, 13.7.1, 17.1–17.4, 17.7, 21.1–21.2, 21.4–21.5, 22.1, and 22.3–22.4. Of course, instructors are not restricted to covering only these sections. Material in other sections and chapters can be included to supplement the CC2001 recommendations.

If the principles-and-survey approach is used, Chapters 2, 6, 10, 14, and 18 should be in the classroom presentation. If the additional material places too great a demand on classroom time, instructors can be selective about which topics from the principles chapters to present. For example, the proof of the optimality of shortest job first and the formalism for name spaces can be safely omitted. Other examples of existing operating systems, not included here, make for good outside reading assignments.

It is also quite common to structure operating systems courses with a “hands-on” component. In this approach, students are generally expected to familiarize themselves with and make modifications to an existing OS. Frequently, the OS that is used is a relatively small one to make the expectations more manageable. This book also provides material that supports this style of course. The instructor can choose between two existing operating systems: Inferno and Linux. Instructors using Inferno should cover Chapters 3, 7, 11, 15, and 19 in addition to the general principles. Similarly for Linux, Chapters 4, 8, 12, 16, and 20 should be included. In covering the chapters that present code, it is important not to present too much code in the classroom. Experience has shown that it doesn’t take long for all code to start looking the same, and the additional benefit dwindles. It is better to present a few smaller bits that illustrate particularly important points and have the students learn the rest with outside reading and assignments. The ability to read and understand real code is a valuable benefit of this type of organization.

The last curriculum example to consider is that of an advanced operating systems course. Considering the nature of advanced courses, several parts of the book could be used well. If the students have come from an introductory course that did not cover all the principles discussed here, then classroom time could be spent presenting them and digging more deeply into any of the principles. Similarly, the survey chapters are a good launching point for a more thorough examination of any one of them or for a more encompassing survey of real operating systems. Finally, for those students whose introductory course did not include experience with operating system internals, the detailed coverage of Inferno and Linux provides a starting place for such experience. These various curriculum designs are summarized in the following table:

Chapter	Principles Only	Principles and Examples	Principles and Inferno	Principles and Linux
1	✓	✓	✓	✓
2	•	✓		
3			✓	
4				✓
5	✓	✓	✓	✓
6	•	✓		
7			✓	
8				✓
9	✓	✓	✓	✓
10	•	✓		
11			✓	
12				✓
13	✓	✓	✓	✓
14	•	✓		
15			✓	
16				✓
17	✓	✓	✓	✓
18	•	✓		
19			✓	
20				✓
21	✓	✓	✓	✓
22	✓	✓	✓	✓

✓ : The chapter is covered.
• : Selected topics from the chapter are covered.

Features of the Book

Each division of the book presents a topic from several perspectives: general principles, survey of applications, and detailed design and implementation of Inferno and Linux. The general-principles chapters include a number of key features. Several techniques are presented in the form of semiformal algorithms that are suitable for implementation. These algorithms are set apart typographically. In a number of cases, techniques are illustrated with detailed examples, also with distinct formatting. Finally, these chapters include a number of historical notes to help establish context. In those chapters that present detailed discussions of Inferno and Linux, I focus on relatively small parts of the kernel that illustrate the techniques and principles covered in the principles chapters. Each function I present is broken down in to small fragments, and I describe each of those in some detail. The result is a detailed study of some key elements of the respective kernels. These chapters also include exercises that ask the student to “get their hands dirty” making changes to Inferno and to Linux. In addition to these general features, here are some of the key topics discussed:

- *Chapter 1*: background, history, organization, bootstrapping, and system calls
- *Chapters 3, 4*: Inferno and Linux history, structure, initialization, and system calls
- *Chapter 5*: process representation, process scheduling, context switching, mutual exclusion, and deadlock
- *Chapters 7, 8*: process representation, creation, and scheduling in Inferno and Linux, including the new $O(1)$ scheduler in Linux
- *Chapter 9*: address translation techniques, variable-sized allocation techniques (including a comparative example), swapping, and paging
- *Chapter 11*: pool/block allocation and garbage collection in Inferno
- *Chapter 12*: zone/slab allocation, page tables, and page faults in Linux
- *Chapter 13*: overview of I/O hardware, techniques for controlling devices, and selected device management techniques
- *Chapter 15*: device driver structure, parallel port driver, keyboard driver, and IDE disk driver in Inferno
- *Chapter 16*: two-half interrupt handler, parallel port driver, and floppy disk driver in Linux
- *Chapter 17*: name spaces, storage management techniques, and journaled file systems
- *Chapter 19*: Inferno file server design, the Styx protocol, and the kfs file system
- *Chapter 20*: the Linux Virtual File System and the EXT3 file system
- *Chapter 21*: basic security techniques and threats, the Orange Book, encryption, and the Multics protection rings
- *Chapter 22*: resource sharing, synchronous operation, clusters, grids, distributed clocks, and election algorithms

Operating System Examples

The two operating systems discussed in detail in this book each provide their own advantages. Inferno is a relatively small operating system, making its details easier to grasp. Inferno is also somewhat unique in that it was designed to run not only as a conventional native operating system, but also as an application running on a host operating system. This hosted capability makes it significantly easier for students to install the OS on their own machines. It also simplifies the process of testing new versions as students debug their assignments. Linux, on the other hand, is a very familiar system—much more so than Inferno. Consequently, studying it provides the student with more directly applicable experience. It also provides examples of some of the more complex techniques not found in Inferno.

The version of Inferno used in this book is the release of May 10, 2007. The most recent distribution of Inferno can be found at the Vita Nuova Web site, <http://www.vitanuova.com>. Current development and recent revision history can be found on Google's code-hosting site at <http://code.google.com/p/inferno-os/>. I discuss the process of compiling and running Inferno in the appendices.

I use version 2.6.18 of the Linux kernel here. The primary site for both current and older versions of the Linux kernel is <http://www.kernel.org>. Two excellent sources of information on building a Linux kernel can be found in the "Linux Kernel HOWTO" by Brian Ward and the "Kernel Rebuild Guide" by Kwan Lowe.

Some of the source code for examples in Chapters 2, 6, 10, 14, and 18 is also available on the Web. The full source code for the CTSS operating system can be found at <http://www.piercefuller.com/library/ctss.html>. Some portions of the source code to Multics can be found at <http://www.multicians.org>. Old versions of UNIX, including the sixth edition, can be found at <http://www.tuhs.org>. The 4.3BSD version of UNIX is being maintained by the International Free Computing Task Force (IFCTF) as 4.3BSD-Quasijarus. The home for this project is <http://ifctfvax.harhan.org/Quasijarus> where these updates, as well as older versions, are all available. The primary resource for TinyOS is <http://www.tinyos.net>. Finally, the home for the Xen virtual machine monitor is <http://www.cl.cam.ac.uk/research/srg/netos/xen>.

Source Code Formatting

The source code fragments in this book are formatted using Knuth and Levy's CWEB system of structured documentation. Keywords and data types are typeset in boldface. Identifiers not in all caps are typeset in italic; identifiers in all caps are typeset in a monospace font. Several of the C language operators are multicharacter sequences that use characters from the ASCII character set. When presented here, some of these are replaced by common mathematical symbols, which in some cases express the meaning more directly. The correspondence between the C operators in ASCII and the symbols typeset in this text are summarized in the following table:

ASCII	Symbol
->	→
NULL	Λ
~	~
^	⊕
==	≡
!=	≠
>=	≥
<=	≤
!	¬
&&	∧
	∨
>>	≫
<<	≪

In addition to these conventions, **CWEB** formats octal and hexadecimal constants differently. An octal constant that would be expressed as `0123` in ASCII is typeset as `°123`, and the hexadecimal constant `0x123` is typeset as `#123`.

It might seem strange that our printed representation doesn't appear in the same form as the compiler input that is typed in a text editor. Although today we don't see this sort of difference much outside of **WEB** and **CWEB**, it has a long tradition. A number of languages allowed variation in the characters used for operators and such because not all installations used the same character sets. Some environments had very limited character sets with some not even including lowercase letters. Others had extremely rich character sets, allowing programmers to directly enter mathematical symbols, such as those we use here for “not equal to,” “less than or equal to,” and “greater than or equal to.” C itself defines trigraphs to allow for its use in environments that do not include all needed characters. Beyond that, it has been quite common for published code to have a different look to it, much as typeset text has a different look from the output of a typewriter. In that spirit, Algol 68 formalized the difference between the published form, called the strict language, and the compiler-input form, called the reference language. It is interesting to note that Algol was one of the languages that influenced the design of C, and the typesetting practices used with Algol influenced Knuth's development of **WEB**. The two lines of influence have converged in **CWEB**, whose printed representation I use here.

Supplementary Material

Supporting material for this book can be found on the Course Technology Web site at <http://www.course.com>. This same material is included on the instructor's CD, which can be obtained from a Course Technology sales representative. Copies of the relevant versions of *Inferno* and *Linux* are included with the supplementary material. The material also includes solutions to most of the exercises. Presentation material is included as well.

Acknowledgments

No project of this magnitude can be completed without help. There are numerous people whose support, encouragement, and assistance have been invaluable. I would first like to thank my wife Mary and my daughter Rachel. They have put up with innumerable nights of my frustration and of losing me to my office. But they never stopped believing in me or in this project. They picked up the slack around the house as I wrote. The interest and support of my other family and friends have also been a much welcome source of encouragement.

Next, I would like to thank my colleagues, both at FedEx and at the University of Memphis. No matter how hard an author tries to manage time (or at least no matter how hard *I* try to manage time), writing a book like this affects other responsibilities. I would particularly like to express my appreciation to Miley Ainsworth of FedEx Labs for his support of this project. I would also like to thank my colleague at FedEx, Tim Gregory, for his assistance in understanding TinyOS and for reviewing what I wrote about it.

The next group of people I'd like to thank is all of my operating systems students over the past few years. They have been my test subjects for this book. They have made

do with partial drafts, typos, and awkward grammar. Nevertheless, many of them have expressed support and encouragement for the project. I'd like to recognize a few students who provided helpful feedback. They include Bob Bradley, Jim Greer, Taliesin Penfound, and Debbie Travis.

I have also corresponded with several members of the operating systems community who provided valuable assistance and feedback. They are Charles Forsyth, of Vita Nuova Holdings Ltd.; Tom Van Vleck, former member of the Multics development team and maintainer of the multicians.org Web site; Stephen Hoffman, of the HP OpenVMS group; and Digby Tarvin, of the University of New South Wales. I would also like to thank Jim Aman at Saint Xavier University, Charles Anderson at Western Oregon University, Bob Bradley at the University of Tennessee at Martin, Thomas Liu at New Jersey City University, Chung-Horng Lung at Carleton University, Jon Weissman at the University of Minnesota, and Dongyan Xu at Purdue University. Their careful review of the early drafts led to substantial improvements in this material.

Finally, I want to express my appreciation to everyone at Course Technology. There's no way to list everyone who has contributed to this project. However, a few names must be recognized. First, I'd like to thank Mary Franz, who first saw the potential for this book and helped me through the early stages of the process. There are three people who have been there working with me all the way through the project. I am supremely grateful to Alyssa Pratt, Jim Markham, and Matt Hutchinson for everything they've done to make this book possible. Everyone's input and involvement have made this a better book. Any remaining flaws are entirely my own. I can't thank everyone enough for all the support in making this a reality.

Brian L. Stuart