

Principles of Computer System Design

An Introduction

Part II
Chapters 7–11

Jerome H. Saltzer

M. Frans Kaashoek

Massachusetts Institute of Technology

Version 5.0

Copyright © 2009 by Jerome H. Saltzer and M. Frans Kaashoek. Some Rights Reserved.

This work is licensed under a  Creative Commons Attribution-Non-commercial-Share Alike 3.0 United States License. For more information on what this license means, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/us/>

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances in which the authors are aware of a claim, the product names appear in initial capital or all capital letters. All trademarks that appear or are otherwise referred to in this work belong to their respective owners.

Suggestions, Comments, Corrections, and Requests to waive license restrictions:
Please send correspondence by electronic mail to:

Saltzer@mit.edu

and

kaashoek@mit.edu

Contents

PART I [In Printed Textbook]

List of Sidebars	xix
Preface	xxvii
Where to Find Part II and other On-line Materials	xxxvii
Acknowledgments	xxxix
Computer System Design Principles	xliii

CHAPTER 1 Systems	1
Overview	2
1.1. Systems and Complexity	3
1.1.1 Common Problems of Systems in Many Fields	3
1.1.2 Systems, Components, Interfaces and Environments	8
1.1.3 Complexity	10
1.2. Sources of Complexity	13
1.2.1 Cascading and Interacting Requirements	13
1.2.2 Maintaining High Utilization	17
1.3. Coping with Complexity I	19
1.3.1 Modularity	19
1.3.2 Abstraction	20
1.3.3 Layering	24
1.3.4 Hierarchy	25
1.3.5 Putting it Back Together: Names Make Connections	26
1.4. Computer Systems are the Same but Different	27
1.4.1 Computer Systems Have no Nearby Bounds on Composition	28
1.4.2 $d(\text{technology})/dt$ is Unprecedented	31
1.5. Coping with Complexity II	35
1.5.1 Why Modularity, Abstraction, Layering, and Hierarchy aren't Enough	36
1.5.2 Iteration	36
1.5.3 Keep it Simple	39
What the Rest of this Book is about	40
Exercises	41

CHAPTER 2 Elements of Computer System Organization	43
Overview	44
2.1. The Three Fundamental Abstractions	45
2.1.1 Memory	45
2.1.2 Interpreters	53
2.1.3 Communication Links	59
2.2. Naming in Computer Systems	60
2.2.1 The Naming Model.	61
2.2.2 Default and Explicit Context References	66
2.2.3 Path Names, Naming Networks, and Recursive Name Resolution ..	71
2.2.4 Multiple Lookup: Searching through Layered Contexts	73
2.2.5 Comparing Names	75
2.2.6 Name Discovery	76
2.3. Organizing Computer Systems with Names and Layers	78
2.3.1 A Hardware Layer: The Bus.	80
2.3.2 A Software Layer: The File Abstraction	87
2.4. Looking Back and Ahead	90
2.5. Case Study: UNIX® File System Layering and Naming	91
2.5.1 Application Programming Interface for the UNIX File System	91
2.5.2 The Block Layer	93
2.5.3 The File Layer	95
2.5.4 The Inode Number Layer	96
2.5.5 The File Name Layer	96
2.5.6 The Path Name Layer	98
2.5.7 Links	99
2.5.8 Renaming	101
2.5.9 The Absolute Path Name Layer	102
2.5.10 The Symbolic Link Layer	104
2.5.11 Implementing the File System API	106
2.5.12 The Shell, Implied Contexts, Search Paths, and Name Discovery ..	110
2.5.13 Suggestions for Further Reading	112
Exercises	112
CHAPTER 3 The Design of Naming Schemes	115
Overview	115
3.1. Considerations in the Design of Naming Schemes	116
3.1.1 Modular Sharing	116

3.1.2 Metadata and Name Overloading	120
3.1.3 Addresses: Names that Locate Objects	122
3.1.4 Generating Unique Names	124
3.1.5 Intended Audience and User-Friendly Names	127
3.1.6 Relative Lifetimes of Names, Values, and Bindings	129
3.1.7 Looking Back and Ahead: Names are a Basic System Component	131
3.2. Case Study: The Uniform Resource Locator (URL)	132
3.2.1 Surfing as a Referential Experience; Name Discovery	132
3.2.2 Interpretation of the URL	133
3.2.3 URL Case Sensitivity	134
3.2.4 Wrong Context References for a Partial URL	135
3.2.5 Overloading of Names in URLs	137
3.3. War Stories: Pathologies in the Use of Names	138
3.3.1 A Name Collision Eliminates Smiling Faces	139
3.3.2 Fragile Names from Overloading, and a Market Solution	139
3.3.3 More Fragile Names from Overloading, with Market Disruption	140
3.3.4 Case-Sensitivity in User-Friendly Names	141
3.3.5 Running Out of Telephone Numbers	142
Exercises	144
CHAPTER 4 Enforcing Modularity with Clients and Services	147
Overview	148
4.1. Client/service organization	149
4.1.1 From soft modularity to enforced modularity	149
4.1.2 Client/service organization	155
4.1.3 Multiple clients and services	163
4.1.4 Trusted intermediaries	163
4.1.5 A simple example service	165
4.2. Communication between client and service	167
4.2.1 Remote procedure call (RPC)	167
4.2.2 RPCs are not identical to procedure calls	169
4.2.3 Communicating through an intermediary	172
4.3. Summary and the road ahead	173
4.4. Case study: The Internet Domain Name System (DNS)	175
4.4.1 Name resolution in DNS	176
4.4.2 Hierarchical name management	180
4.4.3 Other features of DNS	181

4.4.4	Name discovery in DNS	183
4.4.5	Trustworthiness of DNS responses	184
4.5.	Case study: The Network File System (NFS)	184
4.5.1	Naming remote files and directories	185
4.5.2	The NFS remote procedure calls	187
4.5.3	Extending the UNIX file system to support NFS	190
4.5.4	Coherence	192
4.5.5	NFS version 3 and beyond	194
	Exercises	195

CHAPTER 5 Enforcing Modularity with Virtualization 199

	Overview	200
5.1.	Client/Service Organization within a Computer using Virtualization	201
5.1.1	Abstractions for Virtualizing Computers	203
5.1.1.1	Threads	204
5.1.1.2	Virtual Memory	206
5.1.1.3	Bounded Buffer	206
5.1.1.4	Operating System Interface	207
5.1.2	Emulation and Virtual Machines	208
5.1.3	Roadmap: Step-by-Step Virtualization	208
5.2.	Virtual Links using SEND, RECEIVE, and a Bounded Buffer	210
5.2.1	An Interface for SEND and RECEIVE with Bounded Buffers	210
5.2.2	Sequence Coordination with a Bounded Buffer	211
5.2.3	Race Conditions	214
5.2.4	Locks and Before-or-After Actions	218
5.2.5	Deadlock	221
5.2.6	Implementing ACQUIRE and RELEASE	222
5.2.7	Implementing a Before-or-After Action Using the One-Writer Principle	225
5.2.8	Coordination between Synchronous Islands with Asynchronous Connections	228
5.3.	Enforcing Modularity in Memory	230
5.3.1	Enforcing Modularity with Domains	230
5.3.2	Controlled Sharing using Several Domains	231
5.3.3	More Enforced Modularity with Kernel and User Mode	234
5.3.4	Gates and Changing Modes	235
5.3.5	Enforcing Modularity for Bounded Buffers	237

5.3.6 The Kernel	238
5.4. Virtualizing Memory.	242
5.4.1 Virtualizing Addresses.	243
5.4.2 Translating Addresses using a Page Map	245
5.4.3 Virtual Address Spaces	248
5.4.3.1 Primitives for Virtual Address Spaces	248
5.4.3.2 The Kernel and Address Spaces	250
5.4.3.3 Discussion	251
5.4.4 Hardware versus Software and the Translation Look-Aside Buffer.	252
5.4.5 Segments (Advanced Topic)	253
5.5. Virtualizing Processors using Threads	255
5.5.1 Sharing a processor among multiple threads	255
5.5.2 Implementing YIELD.	260
5.5.3 Creating and Terminating Threads	264
5.5.4 Enforcing Modularity with Threads: Preemptive Scheduling	269
5.5.5 Enforcing Modularity with Threads and Address Spaces.	271
5.5.6 Layering Threads	271
5.6. Thread Primitives for Sequence Coordination	273
5.6.1 The Lost Notification Problem.	273
5.6.2 Avoiding the Lost Notification Problem with Eventcounts and Sequencers	275
5.6.3 Implementing AWAIT, ADVANCE, TICKET, and READ (Advanced Topic).	280
5.6.4 Polling, Interrupts, and Sequence coordination.	282
5.7. Case study: Evolution of Enforced Modularity in the Intel x86	284
5.7.1 The early designs: no support for enforced modularity	285
5.7.2 Enforcing Modularity using Segmentation	286
5.7.3 Page-Based Virtual Address Spaces	287
5.7.4 Summary: more evolution	288
5.8. Application: Enforcing Modularity using Virtual Machines	290
5.8.1 Virtual Machine Uses.	290
5.8.2 Implementing Virtual Machines.	291
5.8.3 Virtualizing Example	293
Exercises	294
CHAPTER 6 Performance	299
Overview.	300

6.1. Designing for Performance	300
6.1.1 Performance Metrics	302
6.1.1.1 Capacity, Utilization, Overhead, and Useful Work	302
6.1.1.2 Latency	302
6.1.1.3 Throughput	303
6.1.2 A Systems Approach to Designing for Performance	304
6.1.3 Reducing latency by exploiting workload properties	306
6.1.4 Reducing Latency Using Concurrency	307
6.1.5 Improving Throughput: Concurrency	309
6.1.6 Queuing and Overload	311
6.1.7 Fighting Bottlenecks	313
6.1.7.1 Batching	314
6.1.7.2 Dallying	314
6.1.7.3 Speculation	314
6.1.7.4 Challenges with Batching, Dallying, and Speculation	315
6.1.8 An Example: the I/O bottleneck	316
6.2. Multilevel Memories.	321
6.2.1 Memory Characterization	322
6.2.2 Multilevel Memory Management using Virtual Memory.	323
6.2.3 Adding multilevel memory management to a virtual memory	327
6.2.4 Analyzing Multilevel Memory Systems	331
6.2.5 Locality of reference and working sets	333
6.2.6 Multilevel Memory Management Policies	335
6.2.7 Comparative analysis of different policies	340
6.2.8 Other Page-Removal Algorithms	344
6.2.9 Other aspects of multilevel memory management	346
6.3. Scheduling	347
6.3.1 Scheduling Resources	348
6.3.2 Scheduling metrics	349
6.3.3 Scheduling Policies	352
6.3.3.1 First-Come, First-Served	353
6.3.3.2 Shortest-job-first	354
6.3.3.3 Round-Robin	355
6.3.3.4 Priority Scheduling	357
6.3.3.5 Real-time Schedulers	359

6.3.4 Case study: Scheduling the Disk Arm	360
Exercises	362
About Part II	369
Appendix A: The Binary Classification Trade-off	371
Suggestions for Further Reading	375
Problem Sets for Part I	425
Glossary	475
Index of Concepts	513

Part II [On-Line]

CHAPTER 7 The Network as a System and as a System Component 7-1

Overview	7-2
7.1. Interesting Properties of Networks	7-3
7.1.1 Isochronous and Asynchronous Multiplexing	7-5
7.1.2 Packet Forwarding; Delay	7-9
7.1.3 Buffer Overflow and Discarded Packets	7-12
7.1.4 Duplicate Packets and Duplicate Suppression	7-15
7.1.5 Damaged Packets and Broken Links	7-18
7.1.6 Reordered Delivery	7-19
7.1.7 Summary of Interesting Properties and the Best-Effort Contract	7-20
7.2. Getting Organized: Layers	7-20
7.2.1 Layers	7-23
7.2.2 The Link Layer	7-25
7.2.3 The Network Layer	7-27
7.2.4 The End-to-End Layer	7-28
7.2.5 Additional Layers and the End-to-End Argument	7-30
7.2.6 Mapped and Recursive Applications of the Layered Model	7-32
7.3. The Link Layer	7-34
7.3.1 Transmitting Digital Data in an Analog World	7-34
7.3.2 Framing Frames	7-38
7.3.3 Error Handling	7-40
7.3.4 The Link Layer Interface: Link Protocols and Multiplexing	7-41
7.3.5 Link Properties	7-44

7.4. The Network Layer	7-46
7.4.1 Addressing Interface	7-46
7.4.2 Managing the Forwarding Table: Routing	7-48
7.4.3 Hierarchical Address Assignment and Hierarchical Routing.	7-56
7.4.4 Reporting Network Layer Errors	7-59
7.4.5 Network Address Translation (An Idea That Almost Works)	7-61
7.5. The End-to-End Layer.	7-62
7.5.1 Transport Protocols and Protocol Multiplexing	7-63
7.5.2 Assurance of At-Least-Once Delivery; the Role of Timers	7-67
7.5.3 Assurance of At-Most-Once Delivery: Duplicate Suppression	7-71
7.5.4 Division into Segments and Reassembly of Long Messages	7-73
7.5.5 Assurance of Data Integrity	7-73
7.5.6 End-to-End Performance: Overlapping and Flow Control.	7-75
7.5.6.1 Overlapping Transmissions	7-75
7.5.6.2 Bottlenecks, Flow Control, and Fixed Windows	7-77
7.5.6.3 Sliding Windows and Self-Pacing	7-79
7.5.6.4 Recovery of Lost Data Segments with Windows	7-81
7.5.7 Assurance of Stream Order, and Closing of Connections.	7-82
7.5.8 Assurance of Jitter Control	7-84
7.5.9 Assurance of Authenticity and Privacy.	7-85
7.6. A Network System Design Issue: Congestion Control.	7-86
7.6.1 Managing Shared Resources	7-86
7.6.2 Resource Management in Networks	7-89
7.6.3 Cross-layer Cooperation: Feedback	7-91
7.6.4 Cross-layer Cooperation: Control	7-93
7.6.5 Other Ways of Controlling Congestion in Networks.	7-94
7.6.6 Delay Revisited	7-98
7.7. Wrapping up Networks	7-99
7.8. Case Study: Mapping the Internet to the Ethernet.	7-100
7.8.1 A Brief Overview of Ethernet	7-100
7.8.2 Broadcast Aspects of Ethernet	7-101
7.8.3 Layer Mapping: Attaching Ethernet to a Forwarding Network	7-103
7.8.4 The Address Resolution Protocol.	7-105
7.9. War Stories: Surprises in Protocol Design	7-107
7.9.1 Fixed Timers Lead to Congestion Collapse in NFS	7-107
7.9.2 Autonet Broadcast Storms	7-108
7.9.3 Emergent Phase Synchronization of Periodic Protocols	7-108

7.9.4 Wisconsin Time Server Meltdown	7–109
Exercises	7–111

CHAPTER 8 Fault Tolerance: Reliable Systems from Unreliable Components

8–1

Overview	8–2
8.1. Faults, Failures, and Fault Tolerant Design.	8–3
8.1.1 Faults, Failures, and Modules	8–3
8.1.2 The Fault-Tolerance Design Process	8–6
8.2. Measures of Reliability and Failure Tolerance.	8–8
8.2.1 Availability and Mean Time to Failure	8–8
8.2.2 Reliability Functions.	8–13
8.2.3 Measuring Fault Tolerance	8–16
8.3. Tolerating Active Faults	8–16
8.3.1 Responding to Active Faults	8–16
8.3.2 Fault Tolerance Models.	8–18
8.4. Systematically Applying Redundancy	8–20
8.4.1 Coding: Incremental Redundancy	8–21
8.4.2 Replication: Massive Redundancy.	8–25
8.4.3 Voting	8–26
8.4.4 Repair.	8–31
8.5. Applying Redundancy to Software and Data	8–36
8.5.1 Tolerating Software Faults.	8–36
8.5.2 Tolerating Software (and other) Faults by Separating State	8–37
8.5.3 Durability and Durable Storage	8–39
8.5.4 Magnetic Disk Fault Tolerance	8–40
8.5.4.1 Magnetic Disk Fault Modes	8–41
8.5.4.2 System Faults	8–42
8.5.4.3 Raw Disk Storage.	8–43
8.5.4.4 Fail-Fast Disk Storage.	8–43
8.5.4.5 Careful Disk Storage	8–45
8.5.4.6 Durable Storage: RAID 1	8–46
8.5.4.7 Improving on RAID 1	8–47
8.5.4.8 Detecting Errors Caused by System Crashes.	8–49
8.5.4.9 Still More Threats to Durability	8–49

8.6. Wrapping up Reliability	8-51
8.6.1 Design Strategies and Design Principles	8-51
8.6.2 How about the End-to-End Argument?	8-52
8.6.3 A Caution on the Use of Reliability Calculations	8-53
8.6.4 Where to Learn More about Reliable Systems	8-53
8.7. Application: A Fault Tolerance Model for CMOS RAM	8-55
8.8. War Stories: Fault Tolerant Systems that Failed	8-57
8.8.1 Adventures with Error Correction	8-57
8.8.2 Risks of Rarely-Used Procedures: The National Archives	8-59
8.8.3 Non-independent Replicas and Backhoe Fade	8-60
8.8.4 Human Error May Be the Biggest Risk	8-61
8.8.5 Introducing a Single Point of Failure	8-63
8.8.6 Multiple Failures: The SOHO Mission Interruption	8-63
Exercises	8-64

CHAPTER 9 Atomicity: All-or-Nothing and Before-or-After 9-1

Overview	9-2
9.1. Atomicity	9-4
9.1.1 All-or-Nothing Atomicity in a Database	9-5
9.1.2 All-or-Nothing Atomicity in the Interrupt Interface	9-6
9.1.3 All-or-Nothing Atomicity in a Layered Application	9-8
9.1.4 Some Actions With and Without the All-or-Nothing Property	9-10
9.1.5 Before-or-After Atomicity: Coordinating Concurrent Threads	9-13
9.1.6 Correctness and Serialization	9-16
9.1.7 All-or-Nothing and Before-or-After Atomicity	9-19
9.2. All-or-Nothing Atomicity I: Concepts	9-21
9.2.1 Achieving All-or-Nothing Atomicity: ALL_OR_NOTHING_PUT	9-21
9.2.2 Systematic Atomicity: Commit and the Golden Rule	9-27
9.2.3 Systematic All-or-Nothing Atomicity: Version Histories	9-30
9.2.4 How Version Histories are Used	9-37
9.3. All-or-Nothing Atomicity II: Pragmatics	9-38
9.3.1 Atomicity Logs	9-39
9.3.2 Logging Protocols	9-42
9.3.3 Recovery Procedures	9-45
9.3.4 Other Logging Configurations: Non-Volatile Cell Storage	9-47
9.3.5 Checkpoints	9-51
9.3.6 What if the Cache is not Write-Through? (Advanced Topic)	9-53

9.4. Before-or-After Atomicity I: Concepts	9-54
9.4.1 Achieving Before-or-After Atomicity: Simple Serialization	9-54
9.4.2 The Mark-Point Discipline.	9-58
9.4.3 Optimistic Atomicity: Read-Capture (Advanced Topic)	9-63
9.4.4 Does Anyone Actually Use Version Histories for Before-or-After Atomicity?	9-67
9.5. Before-or-After Atomicity II: Pragmatics	9-69
9.5.1 Locks	9-70
9.5.2 Simple Locking.	9-72
9.5.3 Two-Phase Locking.	9-73
9.5.4 Performance Optimizations	9-75
9.5.5 Deadlock; Making Progress	9-76
9.6. Atomicity across Layers and Multiple Sites.	9-79
9.6.1 Hierarchical Composition of Transactions	9-80
9.6.2 Two-Phase Commit	9-84
9.6.3 Multiple-Site Atomicity: Distributed Two-Phase Commit	9-85
9.6.4 The Dilemma of the Two Generals.	9-90
9.7. A More Complete Model of Disk Failure (Advanced Topic)	9-92
9.7.1 Storage that is Both All-or-Nothing and Durable	9-92
9.8. Case Studies: Machine Language Atomicity	9-95
9.8.1 Complex Instruction Sets: The General Electric 600 Line.	9-95
9.8.2 More Elaborate Instruction Sets: The IBM System/370	9-96
9.8.3 The Apollo Desktop Computer and the Motorola M68000 Microprocessor.	9-97
Exercises	9-98
CHAPTER 10 Consistency	10-1
Overview.	10-2
10.1. Constraints and Interface Consistency	10-2
10.2. Cache Coherence	10-4
10.2.1 Coherence, Replication, and Consistency in a Cache	10-4
10.2.2 Eventual Consistency with Timer Expiration	10-5
10.2.3 Obtaining Strict Consistency with a Fluorescent Marking Pen	10-7
10.2.4 Obtaining Strict Consistency with the Snoopy Cache.	10-7
10.3. Durable Storage Revisited: Widely Separated Replicas	10-9
10.3.1 Durable Storage and the Durability Mantra	10-9
10.3.2 Replicated State Machines	10-11

- 10.3.3 Shortcuts to Meet more Modest Requirements 10–13
- 10.3.4 Maintaining Data Integrity 10–15
- 10.3.5 Replica Reading and Majorities 10–16
- 10.3.6 Backup 10–17
- 10.3.7 Partitioning Data 10–18
- 10.4. Reconciliation 10–19**
 - 10.4.1 Occasionally Connected Operation 10–20
 - 10.4.2 A Reconciliation Procedure 10–22
 - 10.4.3 Improvements 10–25
 - 10.4.4 Clock Coordination 10–26
- 10.5. Perspectives 10–26**
 - 10.5.1 History 10–27
 - 10.5.2 Trade-Offs 10–28
 - 10.5.3 Directions for Further Study 10–31
- Exercises 10–32

CHAPTER 11 Information Security 11–1

- Overview 11–4
- 11.1. Introduction to Secure Systems 11–5**
 - 11.1.1 Threat Classification 11–7
 - 11.1.2 Security is a Negative Goal 11–9
 - 11.1.3 The Safety Net Approach 11–10
 - 11.1.4 Design Principles 11–13
 - 11.1.5 A High d(technology)/dt Poses Challenges For Security 11–17
 - 11.1.6 Security Model 11–18
 - 11.1.7 Trusted Computing Base 11–26
 - 11.1.8 The Road Map for this Chapter 11–28
- 11.2. Authenticating Principals 11–28**
 - 11.2.1 Separating Trust from Authenticating Principals 11–29
 - 11.2.2 Authenticating Principals 11–30
 - 11.2.3 Cryptographic Hash Functions, Computationally Secure, Window of Validity 11–32
 - 11.2.4 Using Cryptographic Hash Functions to Protect Passwords 11–34
- 11.3. Authenticating Messages 11–36**
 - 11.3.1 Message Authentication is Different from Confidentiality 11–37
 - 11.3.2 Closed versus Open Designs and Cryptography 11–38
 - 11.3.3 Key-Based Authentication Model 11–41

11.3.4	Properties of SIGN and VERIFY	11-41
11.3.5	Public-key versus Shared-Secret Authentication	11-44
11.3.6	Key Distribution	11-45
11.3.7	Long-Term Data Integrity with Witnesses	11-48
11.4.	Message Confidentiality	11-49
11.4.1	Message Confidentiality Using Encryption	11-49
11.4.2	Properties of ENCRYPT and DECRYPT	11-50
11.4.3	Achieving both Confidentiality and Authentication	11-52
11.4.4	Can Encryption be Used for Authentication?	11-53
11.5.	Security Protocols	11-54
11.5.1	Example: Key Distribution	11-54
11.5.2	Designing Security Protocols	11-60
11.5.3	Authentication Protocols	11-63
11.5.4	An Incorrect Key Exchange Protocol	11-66
11.5.5	Diffie-Hellman Key Exchange Protocol	11-68
11.5.6	A Key Exchange Protocol Using a Public-Key System	11-69
11.5.7	Summary	11-71
11.6.	Authorization: Controlled Sharing	11-72
11.6.1	Authorization Operations	11-73
11.6.2	The Simple Guard Model	11-73
11.6.2.1	The Ticket System	11-74
11.6.2.2	The List System	11-74
11.6.2.3	Tickets Versus Lists, and Agencies	11-75
11.6.2.4	Protection Groups	11-76
11.6.3	Example: Access Control in UNIX	11-76
11.6.3.1	Principals in UNIX	11-76
11.6.3.2	ACLs in UNIX	11-77
11.6.3.3	The Default Principal and Permissions of a Process	11-78
11.6.3.4	Authenticating Users	11-79
11.6.3.5	Access Control Check	11-79
11.6.3.6	Running Services	11-80
11.6.3.7	Summary of UNIX Access Control	11-80
11.6.4	The Caretaker Model	11-80
11.6.5	Non-Discretionary Access and Information Flow Control	11-81
11.6.5.1	Information Flow Control Example	11-83
11.6.5.2	Covert Channels	11-84

- 11.7. Advanced Topic: Reasoning about Authentication. 11–85**
 - 11.7.1 Authentication Logic. 11–86
 - 11.7.1.1 Hard-wired Approach 11–88
 - 11.7.1.2 Internet Approach. 11–88
 - 11.7.2 Authentication in Distributed Systems 11–89
 - 11.7.3 Authentication across Administrative Realms. 11–90
 - 11.7.4 Authenticating Public Keys 11–92
 - 11.7.5 Authenticating Certificates 11–94
 - 11.7.6 Certificate Chains 11–97
 - 11.7.6.1 Hierarchy of Central Certificate Authorities 11–97
 - 11.7.6.2 Web of Trust 11–98
- 11.8. Cryptography as a Building Block (Advanced Topic). 11–99**
 - 11.8.1 Unbreakable Cipher for Confidentiality (*One-Time Pad*). 11–99
 - 11.8.2 Pseudorandom Number Generators. 11–101
 - 11.8.2.1 Rc4: A Pseudorandom Generator and its Use 11–101
 - 11.8.2.2 Confidentiality using RC4. 11–102
 - 11.8.3 Block Ciphers 11–103
 - 11.8.3.1 Advanced Encryption Standard (AES). 11–103
 - 11.8.3.2 Cipher-Block Chaining. 11–105
 - 11.8.4 Computing a Message Authentication Code 11–106
 - 11.8.4.1 MACs Using Block Cipher or Stream Cipher 11–107
 - 11.8.4.2 MACs Using a Cryptographic Hash Function. 11–107
 - 11.8.5 A Public-Key Cipher 11–109
 - 11.8.5.1 Rivest-Shamir-Adleman (RSA) Cipher 11–109
 - 11.8.5.2 Computing a Digital Signature 11–111
 - 11.8.5.3 A Public-Key Encrypting System. 11–112
- 11.9. Summary 11–112**
- 11.10. Case Study: Transport Layer Security (TLS) for the Web. 11–116**
 - 11.10.1 The TLS Handshake 11–117
 - 11.10.2 Evolution of TLS. 11–120
 - 11.10.3 Authenticating Services with TLS 11–121
 - 11.10.4 User Authentication 11–123
- 11.11. War Stories: Security System Breaches 11–125**
 - 11.11.1 Residues: Profitable Garbage 11–126
 - 11.11.1.1 1963: Residues in CTSS 11–126

- 11.11.1.2 1997: Residues in Network Packets 11–127
- 11.11.1.3 2000: Residues in HTTP 11–127
- 11.11.1.4 Residues on Removed Disks 11–128
- 11.11.1.5 Residues in Backup Copies 11–128
- 11.11.1.6 Magnetic Residues: High-Tech Garbage Analysis 11–129
- 11.11.1.7 2001 and 2002: More Low-tech Garbage Analysis 11–129
- 11.11.2 Plaintext Passwords Lead to Two Breaches 11–130
- 11.11.3 The Multiply Buggy Password Transformation 11–131
- 11.11.4 Controlling the Configuration 11–131
 - 11.11.4.1 Authorized People Sometimes do Unauthorized Things 11–132
 - 11.11.4.2 The System Release Trick 11–132
 - 11.11.4.3 The Slammer Worm 11–132
- 11.11.5 The Kernel Trusts the User 11–135
 - 11.11.5.1 Obvious Trust 11–135
 - 11.11.5.2 Nonobvious Trust (Tocttou) 11–136
 - 11.11.5.3 Tocttou 2: Virtualizing the DMA Channel 11–136
- 11.11.6 Technology Defeats Economic Barriers 11–137
 - 11.11.6.1 An Attack on Our System Would be Too Expensive . . . 11–137
 - 11.11.6.2 Well, it Used to be Too Expensive 11–137
- 11.11.7 Mere Mortals Must be Able to Figure Out How to Use it . . . 11–138
- 11.11.8 The Web can be a Dangerous Place 11–139
- 11.11.9 The Reused Password 11–140
- 11.11.10 Signaling with Clandestine Channels 11–141
 - 11.11.10.1 Intentionally I: Banging on the Walls 11–141
 - 11.11.10.2 Intentionally II 11–141
 - 11.11.10.3 Unintentionally 11–142
- 11.11.11 It Seems to be Working Just Fine 11–142
 - 11.11.11.1 I Thought it was Secure 11–143
 - 11.11.11.2 How Large is the Key Space...Really? 11–144
 - 11.11.11.3 How Long are the Keys? 11–145
- 11.11.12 Injection For Fun and Profit 11–145
 - 11.11.12.1 Injecting a Bogus Alert Message to the Operator 11–146
 - 11.11.12.2 CardSystems Exposes 40,000,000 Credit Card Records to SQL Injection 11–146
- 11.11.13 Hazards of Rarely-Used Components 11–148

11.11.14 A Thorough System Penetration Job	11-148
11.11.15 Framing Enigma	11-149
Exercises	11-151
Suggestions for Further Reading	SR-1
Problem Sets	PS-1
Glossary	GL-1
Complete Index of Concepts	INDEX-1

List of Sidebars

PART I [In Printed Textbook]

CHAPTER 1 Systems

Sidebar 1.1: Stopping a Supertanker	6
Sidebar 1.2: Why Airplanes can't Fly	7
Sidebar 1.3: Terminology: Words used to Describe System Composition	9
Sidebar 1.4: The Cast of Characters and Organizations	14
Sidebar 1.5: How Modularity Reshaped the Computer Industry.	21
Sidebar 1.6: Why Computer Technology has Improved Exponentially with Time.	32

CHAPTER 2 Elements of Computer System Organization

Sidebar 2.1: Terminology: durability, stability, and persistence	46
Sidebar 2.2: How magnetic disks work	49
Sidebar 2.3: Representation: pseudocode and messages	54
Sidebar 2.4: What is an operating system?.	79
Sidebar 2.5: Human engineering and the principle of least astonishment	85

CHAPTER 3 The Design of Naming Schemes

Sidebar 3.1: Generating a unique name from a timestamp	125
Sidebar 3.2: Hypertext links in the Shakespeare Electronic Archive.	129

CHAPTER 4 Enforcing Modularity with Clients and Services

Sidebar 4.1: Enforcing modularity with a high-level languages	154
Sidebar 4.2: Representation: Timing diagrams	156
Sidebar 4.3: Representation: Big-Endian or Little-Endian?	158
Sidebar 4.4: The X Window System	162
Sidebar 4.5: Peer-to-peer: computing without trusted intermediaries	164

CHAPTER 5 Enforcing Modularity with Virtualization

Sidebar 5.1: RSM, test-and-set and avoiding locks	224
Sidebar 5.2: Constructing a before-or-after action without special instructions	226
Sidebar 5.3: Bootstrapping an operating system	239
Sidebar 5.4: Process, thread, and address space	249
Sidebar 5.5: Position-independent programs	251
Sidebar 5.6: Interrupts, exceptions, faults, traps, and signals.	259
Sidebar 5.7: Avoiding the lost notification problem with semaphores	277

CHAPTER 6 Performance

Sidebar 6.1: Design hint: When in doubt use brute force	301
--	-----

Sidebar 6.2: Design hint: Optimiz for the common case 307
Sidebar 6.3: Design hint: Instead of reducing latency, hide it 310
Sidebar 6.4: RAM latency. 323
Sidebar 6.5: Design hint: Separate mechanism from policy. 330
Sidebar 6.6: OPT is a stack algorithm and optimal. 343
Sidebar 6.7: Receive livelock. 350
Sidebar 6.8: Priority inversion 358

Part II [On-Line]

CHAPTER 7 The Network as a System and as a System Component

Sidebar 7.1: Error detection, checksums, and witnesses 7-10
Sidebar 7.2: The Internet 7-32
Sidebar 7.3: Framing phase-encoded bits 7-37
Sidebar 7.4: Shannon's capacity theorem 7-37
Sidebar 7.5: Other end-to-end transport protocol interfaces. 7-66
Sidebar 7.6: Exponentially weighted moving averages. 7-70
Sidebar 7.7: What does an acknowledgment really mean?. 7-77
Sidebar 7.8: The tragedy of the commons 7-93
Sidebar 7.9: Retrofitting TCP. 7-95
Sidebar 7.10: The invisible hand 7-98

CHAPTER 8 Fault Tolerance: Reliable Systems from Unreliable Components

Sidebar 8.1: Reliability functions 8-14
Sidebar 8.2: Risks of manipulating MTTFs 8-30
Sidebar 8.3: Are disk system checksums a wasted effort?. 8-49
Sidebar 8.4: Detecting failures with heartbeats. 8-54

CHAPTER 9 Atomicity: All-or-Nothing and Before-or-After

Sidebar 9.1: Actions and transactions 9-4
Sidebar 9.2: Events that might lead to invoking an exception handler 9-7
Sidebar 9.3: Cascaded aborts 9-29
Sidebar 9.4: The many uses of logs. 9-40

CHAPTER 10 Consistency

CHAPTER 11 Information Security

Sidebar 11.1: Privacy11-7
Sidebar 11.2: Should designs and vulnerabilities be public?11-14
Sidebar 11.3: Malware: viruses, worms, trojan horses, logic bombs, bots, etc...11-19
Sidebar 11.4: Why are buffer overrun bugs so common?11-23
Sidebar 11.5: Authenticating personal devices: the resurrecting duckling policy .11-47
Sidebar 11.6: The Kerberos authentication system11-58
Sidebar 11.7: Secure Hash Algorithm (SHA)11-108
Sidebar 11.8: Economics of computer security11-115

Preface to Part II

This textbook, *Principles of Computer System Design: An Introduction*, is an introduction to the principles and abstractions used in the design of computer systems. It is an outgrowth of notes written by the authors for the M.I.T. Electrical Engineering and Computer Science course 6.033, Computer System Engineering, over a period of 40-plus years.

The book is published in two parts:

- Part I, containing chapters 1-6 and supporting materials for those chapters, is a traditional printed textbook published by Morgan Kaufman, an imprint of Elsevier. (ISBN: 978-012374957-4)
- Part II, consisting of Chapters 7-11 and supporting materials for those chapters, is made available on-line by M.I.T. OpenCourseWare and the authors as an open educational resource.

Availability of the two parts and various supporting materials is described in the section with that title below.

Part II of the textbook continues a main theme of Part I—enforcing modularity—by introducing still stronger forms of modularity. Part I introduces methods that help prevent accidental errors in one module from propagating to another. Part II introduces stronger forms of modularity that can help protect against component and system failures and against malicious attacks. Part II explores communication networks, constructing reliable systems from unreliable components, creating all-or-nothing and before-or-after transactions, and implementing security. In doing so, Part II also continues a second main theme of Part I by introducing several additional design principles related to stronger forms of modularity.

A detailed description of the contents of the chapters of Part II can be found in Part I, in the section “About Part II” on page 369. Part II also includes a table of contents for both Parts I and II, copies of the Suggested Additional Readings and Glossary, Problem Sets for both Parts I and II, and a comprehensive Index of Concepts with page numbers for both Parts I and II in a single alphabetic list.

Availability

The authors and MIT OpenCourseWare provide, free of charge, on-line versions of Chapters 7 through 11, the problem sets, the glossary, and a comprehensive index. Those materials can be found at

<http://ocw.mit.edu/Saltzer-Kaashoek>

in the form of a series of PDF files (requires Adobe Reader), one per chapter or major supporting section, as well as a single PDF file containing the entire set.

The publisher of the printed book also maintains a set of on-line resources at

www.ElsevierDirect.com/9780123749574

Click on the link “Companion Materials”, where you will find Part II of the book as well as other resources, including figures from the text in several formats. Additional materials for instructors (registration required) can be found by clicking the “Manual” link.

There are two additional sources of supporting material related to the teaching of course 6.033 Computer Systems Engineering, at M.I.T. The first source is an OpenCourseWare site containing materials from the teaching of the class in 2005: a class description; lecture, reading, and assignment schedule; board layouts; and many lecture videos. These materials are at

<http://ocw.mit.edu/6-033>

The second source is a Web site for the current 6.033 class. This site contains the current lecture schedule which includes assignments, lecturer notes, and slides. There is also a thirteen-year archive of class assignments, design projects, and quizzes. These materials are all at

<http://mit.edu/6.033>

(Some copyrighted or privacy-sensitive materials on that Web site are restricted to current MIT students.)

Acknowledgments

This textbook began as a set of notes for the advanced undergraduate course Engineering of Computer Systems (6.033, originally 6.233), offered by the Department of Electrical Engineering and Computer Science of the Massachusetts Institute of Technology starting in 1968. The text has benefited from some four decades of comments and suggestions by many faculty members, visitors, recitation instructors, teaching assistants, and students. Over 5,000 students have used (and suffered through) draft versions, and observations of their learning experiences (as well as frequent confusion caused by the text) have informed the writing. We are grateful for those many contributions. In addition, certain aspects deserve specific acknowledgment.

1. Naming (Section 2.2 and Chapter 3)

The concept and organization of the materials on naming grew out of extensive discussions with Michael D. Schroeder. The naming model (and part of our development) follows closely the one developed by D. Austin Henderson in his Ph.D. thesis. Stephen A. Ward suggested some useful generalizations of the naming model, and Roger Needham suggested several concepts in response to an earlier version of this material. That earlier version, including in-depth examples of the naming model applied to addressing architectures and file systems, and an historical bibliography, was published as Chapter 3 in Rudolf Bayer et al., editors, *Operating Systems: An Advanced Course, Lecture Notes in Computer Science 60*, pages 99–208. Springer-Verlag, 1978, reprinted 1984. Additional ideas have been contributed by many others, including Ion Stoica, Karen Solins, Daniel Jackson, Butler Lampson, David Karger, and Hari Balakrishnan.

2. Enforced Modularity and Virtualization (Chapters 4 and 5)

Chapter 4 was heavily influenced by lectures on the same topic by David L. Tennenhouse. Both chapters have been improved by substantial feedback from Hari Balakrishnan, Russ Cox, Michael Ernst, Eddie Kohler, Chris Laas, Barbara H. Liskov, Nancy Lynch, Samuel Madden, Robert T. Morris, Max Poletto, Martin Rinard, Susan Ruff, Gerald Jay Sussman, Julie Sussman, and Michael Walfish.

3. Networks (Chapter 7[on-line])

Conversations with David D. Clark and David L. Tennenhouse were instrumental in laying out the organization of this chapter, and lectures by Clark were the basis for part of the presentation. Robert H. Halstead Jr. wrote an early draft set of notes about networking, and some of his ideas have also been borrowed. Hari Balakrishnan provided many suggestions and corrections and helped sort out muddled explanations, and Julie Sussman and Susan Ruff pointed out many opportunities to improve the presentation. The material on congestion control was developed with the help of extensive discussions

xxv

with Hari Balakrishnan and Robert T. Morris, and is based in part on ideas from Raj Jain.

4. Fault Tolerance (Chapter 8[on-line])

Most of the concepts and examples in this chapter were originally articulated by Claude Shannon, Edward F. Moore, David Huffman, Edward J. McCluskey, Butler W. Lampson, Daniel P. Siewiorek, and Jim N. Gray.

5. Transactions and Consistency (Chapters 9[on-line] and 10[on-line])

The material of the transactions and consistency chapters has been developed over the course of four decades with aid and ideas from many sources. The concept of version histories is due to Jack Dennis, and the particular form of all-or-nothing and before-or-after atomicity with version histories developed here is due to David P. Reed. Jim N. Gray not only came up with many of the ideas described in these two chapters, he also provided extensive comments. (That doesn't imply endorsement—he disagreed strongly about the importance of some of the ideas!) Other helpful comments and suggestions were made by Hari Balakrishnan, Andrew Herbert, Butler W. Lampson, Barbara H. Liskov, Samuel R. Madden, Larry Rudolph, Gerald Jay Sussman, and Julie Sussman.

6. Computer Security (Chapter 11[on-line])

Sections 11.1 and 11.6 draw heavily from the paper “The Protection of Information in Computer Systems” by Jerome H. Saltzer and Michael D. Schroeder, *Proceedings of the IEEE* 63, 9 (September, 1975), pages 1278–1308. Ronald Rivest, David Mazières, and Robert T. Morris made significant contributions to material presented throughout the chapter. Brad Chen, Michael Ernst, Kevin Fu, Charles Leiserson, Susan Ruff, and Seth Teller made numerous suggestions for improving the text.

7. Suggested Outside Readings

Ideas for suggested readings have come from many sources. Particular thanks must go to Michael D. Schroeder, who uncovered several of the classic systems papers in places outside computer science where nobody else would have thought to look, Edward D. Lazowska, who provided an extensive reading list used at the University of Washington, and Butler W. Lampson, who provided a thoughtful review of the list.

8. The Exercises and Problem Sets

The exercises at the end of each chapter and the problem sets at the end of the book have been collected, suggested, tried, debugged, and revised by many different faculty members, instructors, teaching assistants, and undergraduate students over a period of 40 years in the process of constructing quizzes and examinations while teaching the material of the text.

Certain of the longer exercises and most of the problem sets, which are based on lead-in stories and include several related questions, represent a substantial effort by a single individual. For those problem sets not developed by one of the authors, a credit line appears in a footnote on the first page of the problem set.

Following each problem or problem set is an identifier of the form “1978–3–14”. This identifier reports the year, examination number, and problem number of the examination in which some version of that problem first appeared.

Jerome H. Saltzer
M. Frans Kaashoek
2009

Computer System Design Principles

Throughout the text, the description of a design principle presents its name in a **bold-faced** display, and each place that the principle is used highlights it in *underlined italics*.

Design principles applicable to many areas of computer systems

- **Adopt sweeping simplifications**
So you can see what you are doing.
- **Avoid excessive generality**
If it is good for everything, it is good for nothing.
- **Avoid rarely used components**
Deterioration and corruption accumulate unnoticed—until the next use.
- **Be explicit**
Get all of the assumptions out on the table.
- **Decouple modules with indirection**
Indirection supports replaceability.
- **Design for iteration**
You won't get it right the first time, so make it easy to change.
- **End-to-end argument**
The application knows best.
- **Escalating complexity principle**
Adding a feature increases complexity out of proportion.
- **Incommensurate scaling rule**
Changing a parameter by a factor of ten requires a new design.
- **Keep digging principle**
Complex systems fail for complex reasons.
- **Law of diminishing returns**
The more one improves some measure of goodness, the more effort the next improvement will require.
- **Open design principle**
Let anyone comment on the design; you need all the help you can get.
- **Principle of least astonishment**
People are part of the system. Choose interfaces that match the user's experience,

xxix

expectations, and mental models.

- **Robustness principle**
Be tolerant of inputs, strict on outputs.
- **Safety margin principle**
Keep track of the distance to the edge of the cliff or you may fall over the edge.
- **Unyielding foundations rule**
It is easier to change a module than to change the modularity.

Design principles applicable to specific areas of computer systems

- **Atomicity: Golden rule of atomicity**
Never modify the only copy!
- **Coordination: One-writer principle**
If each variable has only one writer, coordination is simpler.
- **Durability: The durability mantra**
Multiple copies, widely separated and independently administered.
- **Security: Minimize secrets**
Because they probably won't remain secret for long.
- **Security: Complete mediation**
Check every operation for authenticity, integrity, and authorization.
- **Security: Fail-safe defaults**
Most users won't change them, so set defaults to do something safe.
- **Security: Least privilege principle**
Don't store lunch in the safe with the jewels.
- **Security: Economy of mechanism**
The less there is, the more likely you will get it right.
- **Security: Minimize common mechanism**
Shared mechanisms provide unwanted communication paths.

Design Hints (useful but not as compelling as design principles)

- Exploit brute force
- Instead of reducing latency, hide it
- Optimize for the common case
- Separate mechanism from policy