

Contents

Preface xix

About the Authors xxxv

1

A Tour of Computer Systems 1

- 1.1 Information Is Bits + Context 3
- 1.2 Programs Are Translated by Other Programs into Different Forms 4
- 1.3 It Pays to Understand How Compilation Systems Work 6
- 1.4 Processors Read and Interpret Instructions Stored in Memory 7
 - 1.4.1 Hardware Organization of a System 8
 - 1.4.2 Running the hello Program 10
- 1.5 Caches Matter 11
- 1.6 Storage Devices Form a Hierarchy 14
- 1.7 The Operating System Manages the Hardware 14
 - 1.7.1 Processes 15
 - 1.7.2 Threads 17
 - 1.7.3 Virtual Memory 18
 - 1.7.4 Files 19
- 1.8 Systems Communicate with Other Systems Using Networks 19
- 1.9 Important Themes 22
 - 1.9.1 Amdahl's Law 22
 - 1.9.2 Concurrency and Parallelism 24
 - 1.9.3 The Importance of Abstractions in Computer Systems 26
- 1.10 Summary 27
 - Bibliographic Notes 28
 - Solutions to Practice Problems 28

Part I Program Structure and Execution

2

Representing and Manipulating Information 31

- 2.1 Information Storage 34
 - 2.1.1 Hexadecimal Notation 36
 - 2.1.2 Data Sizes 39

2.1.3	Addressing and Byte Ordering	42
2.1.4	Representing Strings	49
2.1.5	Representing Code	49
2.1.6	Introduction to Boolean Algebra	50
2.1.7	Bit-Level Operations in C	54
2.1.8	Logical Operations in C	56
2.1.9	Shift Operations in C	57
2.2	Integer Representations	59
2.2.1	Integral Data Types	60
2.2.2	Unsigned Encodings	62
2.2.3	Two's-Complement Encodings	64
2.2.4	Conversions between Signed and Unsigned	70
2.2.5	Signed versus Unsigned in C	74
2.2.6	Expanding the Bit Representation of a Number	76
2.2.7	Truncating Numbers	81
2.2.8	Advice on Signed versus Unsigned	83
2.3	Integer Arithmetic	84
2.3.1	Unsigned Addition	84
2.3.2	Two's-Complement Addition	90
2.3.3	Two's-Complement Negation	95
2.3.4	Unsigned Multiplication	96
2.3.5	Two's-Complement Multiplication	97
2.3.6	Multiplying by Constants	101
2.3.7	Dividing by Powers of 2	103
2.3.8	Final Thoughts on Integer Arithmetic	107
2.4	Floating Point	108
2.4.1	Fractional Binary Numbers	109
2.4.2	IEEE Floating-Point Representation	112
2.4.3	Example Numbers	115
2.4.4	Rounding	120
2.4.5	Floating-Point Operations	122
2.4.6	Floating Point in C	124
2.5	Summary	126
	Bibliographic Notes	127
	Homework Problems	128
	Solutions to Practice Problems	143

3

Machine-Level Representation of Programs 163

3.1 A Historical Perspective 166

3.2	Program Encodings	169
3.2.1	Machine-Level Code	170
3.2.2	Code Examples	172
3.2.3	Notes on Formatting	175
3.3	Data Formats	177
3.4	Accessing Information	179
3.4.1	Operand Specifiers	180
3.4.2	Data Movement Instructions	182
3.4.3	Data Movement Example	186
3.4.4	Pushing and Popping Stack Data	189
3.5	Arithmetic and Logical Operations	191
3.5.1	Load Effective Address	191
3.5.2	Unary and Binary Operations	194
3.5.3	Shift Operations	194
3.5.4	Discussion	196
3.5.5	Special Arithmetic Operations	197
3.6	Control	200
3.6.1	Condition Codes	201
3.6.2	Accessing the Condition Codes	202
3.6.3	Jump Instructions	205
3.6.4	Jump Instruction Encodings	207
3.6.5	Implementing Conditional Branches with Conditional-Control	209
3.6.6	Implementing Conditional Branches with Conditional Moves	214
3.6.7	Loops	220
3.6.8	Switch Statements	232
3.7	Procedures	238
3.7.1	The Run-Time Stack	239
3.7.2	Control Transfer	241
3.7.3	Data Transfer	245
3.7.4	Local Storage on the Stack	248
3.7.5	Local Storage in Registers	251
3.7.6	Recursive Procedures	253
3.8	Array Allocation and Access	255
3.8.1	Basic Principles	255
3.8.2	Pointer Arithmetic	257
3.8.3	Nested Arrays	258
3.8.4	Fixed-Size Arrays	260
3.8.5	Variable-Size Arrays	262

3.9	Heterogeneous Data Structures	265
3.9.1	Structures	265
3.9.2	Unions	269
3.9.3	Data Alignment	273
3.10	Combining Control and Data in Machine-Level Programs	276
3.10.1	Understanding Pointers	277
3.10.2	Life in the Real World: Using the gdb Debugger	279
3.10.3	Out-of-Bounds Memory References and Buffer Overflow	279
3.10.4	Thwarting Buffer Overflow Attacks	284
3.10.5	Supporting Variable-Size Stack Frames	290
3.11	Floating-Point Code	293
3.11.1	Floating-Point Movement and Conversion Operations	296
3.11.2	Floating-Point Code in Procedures	301
3.11.3	Floating-Point Arithmetic Operations	302
3.11.4	Defining and Using Floating-Point Constants	304
3.11.5	Using Bitwise Operations in Floating-Point Code	305
3.11.6	Floating-Point Comparison Operations	306
3.11.7	Observations about Floating-Point Code	309
3.12	Summary	309
	Bibliographic Notes	310
	Homework Problems	311
	Solutions to Practice Problems	325

4

Processor Architecture 351

4.1	The Y86-64 Instruction Set Architecture	355
4.1.1	Programmer-Visible State	355
4.1.2	Y86-64 Instructions	356
4.1.3	Instruction Encoding	358
4.1.4	Y86-64 Exceptions	363
4.1.5	Y86-64 Programs	364
4.1.6	Some Y86-64 Instruction Details	370
4.2	Logic Design and the Hardware Control Language HCL	372
4.2.1	Logic Gates	373
4.2.2	Combinational Circuits and HCL Boolean Expressions	374
4.2.3	Word-Level Combinational Circuits and HCL Integer Expressions	376
4.2.4	Set Membership	380
4.2.5	Memory and Clocking	381
4.3	Sequential Y86-64 Implementations	384
4.3.1	Organizing Processing into Stages	384

4.3.2	SEQ Hardware Structure	396
4.3.3	SEQ Timing	400
4.3.4	SEQ Stage Implementations	404
4.4	General Principles of Pipelining	412
4.4.1	Computational Pipelines	412
4.4.2	A Detailed Look at Pipeline Operation	414
4.4.3	Limitations of Pipelining	416
4.4.4	Pipelining a System with Feedback	419
4.5	Pipelined Y86-64 Implementations	421
4.5.1	SEQ+: Rearranging the Computation Stages	421
4.5.2	Inserting Pipeline Registers	422
4.5.3	Rearranging and Relabeling Signals	426
4.5.4	Next PC Prediction	427
4.5.5	Pipeline Hazards	429
4.5.6	Exception Handling	444
4.5.7	PIPE Stage Implementations	447
4.5.8	Pipeline Control Logic	455
4.5.9	Performance Analysis	464
4.5.10	Unfinished Business	468
4.6	Summary	470
4.6.1	Y86-64 Simulators	472
	Bibliographic Notes	473
	Homework Problems	473
	Solutions to Practice Problems	480

5

Optimizing Program Performance 495

5.1	Capabilities and Limitations of Optimizing Compilers	498
5.2	Expressing Program Performance	502
5.3	Program Example	504
5.4	Eliminating Loop Inefficiencies	508
5.5	Reducing Procedure Calls	512
5.6	Eliminating Unneeded Memory References	514
5.7	Understanding Modern Processors	517
5.7.1	Overall Operation	518
5.7.2	Functional Unit Performance	523
5.7.3	An Abstract Model of Processor Operation	525
5.8	Loop Unrolling	531
5.9	Enhancing Parallelism	536
5.9.1	Multiple Accumulators	536
5.9.2	Reassociation Transformation	541

5.10	Summary of Results for Optimizing Combining Code	547
5.11	Some Limiting Factors	548
5.11.1	Register Spilling	548
5.11.2	Branch Prediction and Misprediction Penalties	549
5.12	Understanding Memory Performance	553
5.12.1	Load Performance	554
5.12.2	Store Performance	555
5.13	Life in the Real World: Performance Improvement Techniques	561
5.14	Identifying and Eliminating Performance Bottlenecks	562
5.14.1	Program Profiling	562
5.14.2	Using a Profiler to Guide Optimization	565
5.15	Summary	568
	Bibliographic Notes	569
	Homework Problems	570
	Solutions to Practice Problems	573

6

The Memory Hierarchy 579

6.1	Storage Technologies	581
6.1.1	Random Access Memory	581
6.1.2	Disk Storage	589
6.1.3	Solid State Disks	600
6.1.4	Storage Technology Trends	602
6.2	Locality	604
6.2.1	Locality of References to Program Data	606
6.2.2	Locality of Instruction Fetches	607
6.2.3	Summary of Locality	608
6.3	The Memory Hierarchy	609
6.3.1	Caching in the Memory Hierarchy	610
6.3.2	Summary of Memory Hierarchy Concepts	614
6.4	Cache Memories	614
6.4.1	Generic Cache Memory Organization	615
6.4.2	Direct-Mapped Caches	617
6.4.3	Set Associative Caches	624
6.4.4	Fully Associative Caches	626
6.4.5	Issues with Writes	630
6.4.6	Anatomy of a Real Cache Hierarchy	631
6.4.7	Performance Impact of Cache Parameters	631
6.5	Writing Cache-Friendly Code	633
6.6	Putting It Together: The Impact of Caches on Program Performance	639

6.6.1	The Memory Mountain	639
6.6.2	Rearranging Loops to Increase Spatial Locality	643
6.6.3	Exploiting Locality in Your Programs	647
6.7	Summary	648
	Bibliographic Notes	648
	Homework Problems	649
	Solutions to Practice Problems	660

Part II Running Programs on a System

7

Linking 669

7.1	Compiler Drivers	671
7.2	Static Linking	672
7.3	Object Files	673
7.4	Relocatable Object Files	674
7.5	Symbols and Symbol Tables	675
7.6	Symbol Resolution	679
7.6.1	How Linkers Resolve Duplicate Symbol Names	680
7.6.2	Linking with Static Libraries	684
7.6.3	How Linkers Use Static Libraries to Resolve References	688
7.7	Relocation	689
7.7.1	Relocation Entries	690
7.7.2	Relocating Symbol References	691
7.8	Executable Object Files	695
7.9	Loading Executable Object Files	697
7.10	Dynamic Linking with Shared Libraries	698
7.11	Loading and Linking Shared Libraries from Applications	701
7.12	Position-Independent Code (PIC)	704
7.13	Library Interpositioning	707
7.13.1	Compile-Time Interpositioning	708
7.13.2	Link-Time Interpositioning	708
7.13.3	Run-Time Interpositioning	710
7.14	Tools for Manipulating Object Files	713
7.15	Summary	713
	Bibliographic Notes	714
	Homework Problems	714
	Solutions to Practice Problems	717

8

Exceptional Control Flow 721

- 8.1 Exceptions 723
 - 8.1.1 Exception Handling 724
 - 8.1.2 Classes of Exceptions 726
 - 8.1.3 Exceptions in Linux/x86-64 Systems 729
- 8.2 Processes 732
 - 8.2.1 Logical Control Flow 732
 - 8.2.2 Concurrent Flows 733
 - 8.2.3 Private Address Space 734
 - 8.2.4 User and Kernel Modes 734
 - 8.2.5 Context Switches 736
- 8.3 System Call Error Handling 737
- 8.4 Process Control 738
 - 8.4.1 Obtaining Process IDs 739
 - 8.4.2 Creating and Terminating Processes 739
 - 8.4.3 Reaping Child Processes 743
 - 8.4.4 Putting Processes to Sleep 749
 - 8.4.5 Loading and Running Programs 750
 - 8.4.6 Using fork and execve to Run Programs 753
- 8.5 Signals 756
 - 8.5.1 Signal Terminology 758
 - 8.5.2 Sending Signals 759
 - 8.5.3 Receiving Signals 762
 - 8.5.4 Blocking and Unblocking Signals 764
 - 8.5.5 Writing Signal Handlers 766
 - 8.5.6 Synchronizing Flows to Avoid Nasty Concurrency Bugs 776
 - 8.5.7 Explicitly Waiting for Signals 778
- 8.6 Nonlocal Jumps 781
- 8.7 Tools for Manipulating Processes 786
- 8.8 Summary 787
 - Bibliographic Notes 787
 - Homework Problems 788
 - Solutions to Practice Problems 795

9

Virtual Memory 801

- 9.1 Physical and Virtual Addressing 803
- 9.2 Address Spaces 804

9.3	VM as a Tool for Caching	805
9.3.1	DRAM Cache Organization	806
9.3.2	Page Tables	806
9.3.3	Page Hits	808
9.3.4	Page Faults	808
9.3.5	Allocating Pages	810
9.3.6	Locality to the Rescue Again	810
9.4	VM as a Tool for Memory Management	811
9.5	VM as a Tool for Memory Protection	812
9.6	Address Translation	813
9.6.1	Integrating Caches and VM	817
9.6.2	Speeding Up Address Translation with a TLB	817
9.6.3	Multi-Level Page Tables	819
9.6.4	Putting It Together: End-to-End Address Translation	821
9.7	Case Study: The Intel Core i7/Linux Memory System	825
9.7.1	Core i7 Address Translation	826
9.7.2	Linux Virtual Memory System	828
9.8	Memory Mapping	833
9.8.1	Shared Objects Revisited	833
9.8.2	The fork Function Revisited	836
9.8.3	The execve Function Revisited	836
9.8.4	User-Level Memory Mapping with the mmap Function	837
9.9	Dynamic Memory Allocation	839
9.9.1	The malloc and free Functions	840
9.9.2	Why Dynamic Memory Allocation?	843
9.9.3	Allocator Requirements and Goals	844
9.9.4	Fragmentation	846
9.9.5	Implementation Issues	846
9.9.6	Implicit Free Lists	847
9.9.7	Placing Allocated Blocks	849
9.9.8	Splitting Free Blocks	849
9.9.9	Getting Additional Heap Memory	850
9.9.10	Coalescing Free Blocks	850
9.9.11	Coalescing with Boundary Tags	851
9.9.12	Putting It Together: Implementing a Simple Allocator	854
9.9.13	Explicit Free Lists	862
9.9.14	Segregated Free Lists	863
9.10	Garbage Collection	865
9.10.1	Garbage Collector Basics	866
9.10.2	Mark&Sweep Garbage Collectors	867
9.10.3	Conservative Mark&Sweep for C Programs	869

9.11	Common Memory-Related Bugs in C Programs	870
9.11.1	Dereferencing Bad Pointers	870
9.11.2	Reading Uninitialized Memory	871
9.11.3	Allowing Stack Buffer Overflows	871
9.11.4	Assuming That Pointers and the Objects They Point to Are the Same Size	872
9.11.5	Making Off-by-One Errors	872
9.11.6	Referencing a Pointer Instead of the Object It Points To	873
9.11.7	Misunderstanding Pointer Arithmetic	873
9.11.8	Referencing Nonexistent Variables	874
9.11.9	Referencing Data in Free Heap Blocks	874
9.11.10	Introducing Memory Leaks	875
9.12	Summary	875
	Bibliographic Notes	876
	Homework Problems	876
	Solutions to Practice Problems	880

Part III Interaction and Communication between Programs

10

System-Level I/O 889

10.1	Unix I/O	890
10.2	Files	891
10.3	Opening and Closing Files	893
10.4	Reading and Writing Files	895
10.5	Robust Reading and Writing with the RIo Package	897
10.5.1	RIo Unbuffered Input and Output Functions	897
10.5.2	RIo Buffered Input Functions	898
10.6	Reading File Metadata	903
10.7	Reading Directory Contents	905
10.8	Sharing Files	906
10.9	I/O Redirection	909
10.10	Standard I/O	911
10.11	Putting It Together: Which I/O Functions Should I Use?	911
10.12	Summary	913
	Bibliographic Notes	914
	Homework Problems	914
	Solutions to Practice Problems	915

11

Network Programming 917

- 11.1** The Client-Server Programming Model 918
- 11.2** Networks 919
- 11.3** The Global IP Internet 924
 - 11.3.1 IP Addresses 925
 - 11.3.2 Internet Domain Names 927
 - 11.3.3 Internet Connections 929
- 11.4** The Sockets Interface 932
 - 11.4.1 Socket Address Structures 933
 - 11.4.2 The socket Function 934
 - 11.4.3 The connect Function 934
 - 11.4.4 The bind Function 935
 - 11.4.5 The listen Function 935
 - 11.4.6 The accept Function 936
 - 11.4.7 Host and Service Conversion 937
 - 11.4.8 Helper Functions for the Sockets Interface 942
 - 11.4.9 Example Echo Client and Server 944
- 11.5** Web Servers 948
 - 11.5.1 Web Basics 948
 - 11.5.2 Web Content 949
 - 11.5.3 HTTP Transactions 950
 - 11.5.4 Serving Dynamic Content 953
- 11.6** Putting It Together: The TINY Web Server 956
- 11.7** Summary 964
 - Bibliographic Notes 965
 - Homework Problems 965
 - Solutions to Practice Problems 966

12

Concurrent Programming 971

- 12.1** Concurrent Programming with Processes 973
 - 12.1.1 A Concurrent Server Based on Processes 974
 - 12.1.2 Pros and Cons of Processes 975
- 12.2** Concurrent Programming with I/O Multiplexing 977
 - 12.2.1 A Concurrent Event-Driven Server Based on I/O Multiplexing 980
 - 12.2.2 Pros and Cons of I/O Multiplexing 985
- 12.3** Concurrent Programming with Threads 985
 - 12.3.1 Thread Execution Model 986

12.3.2	Posix Threads	987
12.3.3	Creating Threads	988
12.3.4	Terminating Threads	988
12.3.5	Reaping Terminated Threads	989
12.3.6	Detaching Threads	989
12.3.7	Initializing Threads	990
12.3.8	A Concurrent Server Based on Threads	991
12.4	Shared Variables in Threaded Programs	992
12.4.1	Threads Memory Model	993
12.4.2	Mapping Variables to Memory	994
12.4.3	Shared Variables	995
12.5	Synchronizing Threads with Semaphores	995
12.5.1	Progress Graphs	999
12.5.2	Semaphores	1001
12.5.3	Using Semaphores for Mutual Exclusion	1002
12.5.4	Using Semaphores to Schedule Shared Resources	1004
12.5.5	Putting It Together: A Concurrent Server Based on Prethreading	1008
12.6	Using Threads for Parallelism	1013
12.7	Other Concurrency Issues	1020
12.7.1	Thread Safety	1020
12.7.2	Reentrancy	1023
12.7.3	Using Existing Library Functions in Threaded Programs	1024
12.7.4	Races	1025
12.7.5	Deadlocks	1027
12.8	Summary	1030
	Bibliographic Notes	1030
	Homework Problems	1031
	Solutions to Practice Problems	1036

A

Error Handling 1041

- A.1** Error Handling in Unix Systems 1042
- A.2** Error-Handling Wrappers 1043

References 1047

Index 1053