

Computer Systems

A Programmer's Perspective

Randal E. Bryant

Carnegie Mellon University

David R. O'Hallaron

Carnegie Mellon University and Intel Labs

Prentice Hall

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Contents

Preface xix

About the Authors xxxiii

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 7
 - 1.4.2 Running the `hello` Program 10
- 1.5** Caches Matter 12
- 1.6** Storage Devices Form a Hierarchy 13
- 1.7** The Operating System Manages the Hardware 14
 - 1.7.1 Processes 16
 - 1.7.2 Threads 17
 - 1.7.3 Virtual Memory 17
 - 1.7.4 Files 19
- 1.8** Systems Communicate with Other Systems Using Networks 20
- 1.9** Important Themes 21
 - 1.9.1 Concurrency and Parallelism 21
 - 1.9.2 The Importance of Abstractions in Computer Systems 24
- 1.10** Summary 25
 - Bibliographic Notes 26

Part I Program Structure and Execution

2

Representing and Manipulating Information 29

- 2.1** Information Storage 33
 - 2.1.1 Hexadecimal Notation 34
 - 2.1.2 Words 38
 - 2.1.3 Data Sizes 38

2.1.4	Addressing and Byte Ordering	39
2.1.5	Representing Strings	46
2.1.6	Representing Code	47
2.1.7	Introduction to Boolean Algebra	48
2.1.8	Bit-Level Operations in C	51
2.1.9	Logical Operations in C	54
2.1.10	Shift Operations in C	54
2.2	Integer Representations	56
2.2.1	Integral Data Types	57
2.2.2	Unsigned Encodings	58
2.2.3	Two's-Complement Encodings	60
2.2.4	Conversions Between Signed and Unsigned	65
2.2.5	Signed vs. Unsigned in C	69
2.2.6	Expanding the Bit Representation of a Number	71
2.2.7	Truncating Numbers	75
2.2.8	Advice on Signed vs. Unsigned	76
2.3	Integer Arithmetic	79
2.3.1	Unsigned Addition	79
2.3.2	Two's-Complement Addition	83
2.3.3	Two's-Complement Negation	87
2.3.4	Unsigned Multiplication	88
2.3.5	Two's-Complement Multiplication	89
2.3.6	Multiplying by Constants	92
2.3.7	Dividing by Powers of Two	95
2.3.8	Final Thoughts on Integer Arithmetic	98
2.4	Floating Point	99
2.4.1	Fractional Binary Numbers	100
2.4.2	IEEE Floating-Point Representation	103
2.4.3	Example Numbers	105
2.4.4	Rounding	110
2.4.5	Floating-Point Operations	113
2.4.6	Floating Point in C	114
2.5	Summary	118
	Bibliographic Notes	119
	Homework Problems	119
	Solutions to Practice Problems	134

3

Machine-Level Representation of Programs 153

3.1	A Historical Perspective	156
3.2	Program Encodings	159

- 3.2.1 Machine-Level Code 160
- 3.2.2 Code Examples 162
- 3.2.3 Notes on Formatting 165
- 3.3** Data Formats 167
- 3.4** Accessing Information 168
 - 3.4.1 Operand Specifiers 169
 - 3.4.2 Data Movement Instructions 171
 - 3.4.3 Data Movement Example 174
- 3.5** Arithmetic and Logical Operations 177
 - 3.5.1 Load Effective Address 177
 - 3.5.2 Unary and Binary Operations 178
 - 3.5.3 Shift Operations 179
 - 3.5.4 Discussion 180
 - 3.5.5 Special Arithmetic Operations 182
- 3.6** Control 185
 - 3.6.1 Condition Codes 185
 - 3.6.2 Accessing the Condition Codes 187
 - 3.6.3 Jump Instructions and Their Encodings 189
 - 3.6.4 Translating Conditional Branches 193
 - 3.6.5 Loops 197
 - 3.6.6 Conditional Move Instructions 206
 - 3.6.7 Switch Statements 213
- 3.7** Procedures 219
 - 3.7.1 Stack Frame Structure 219
 - 3.7.2 Transferring Control 221
 - 3.7.3 Register Usage Conventions 223
 - 3.7.4 Procedure Example 224
 - 3.7.5 Recursive Procedures 229
- 3.8** Array Allocation and Access 232
 - 3.8.1 Basic Principles 232
 - 3.8.2 Pointer Arithmetic 233
 - 3.8.3 Nested Arrays 235
 - 3.8.4 Fixed-Size Arrays 237
 - 3.8.5 Variable-Size Arrays 238
- 3.9** Heterogeneous Data Structures 241
 - 3.9.1 Structures 241
 - 3.9.2 Unions 244
 - 3.9.3 Data Alignment 248
- 3.10** Putting It Together: Understanding Pointers 252
- 3.11** Life in the Real World: Using the GDB Debugger 254
- 3.12** Out-of-Bounds Memory References and Buffer Overflow 256
 - 3.12.1 Thwarting Buffer Overflow Attacks 261

- 3.13** x86-64: Extending IA32 to 64 Bits 267
 - 3.13.1 History and Motivation for x86-64 268
 - 3.13.2 An Overview of x86-64 270
 - 3.13.3 Accessing Information 273
 - 3.13.4 Control 279
 - 3.13.5 Data Structures 290
 - 3.13.6 Concluding Observations about x86-64 291
- 3.14** Machine-Level Representations of Floating-Point Programs 292
- 3.15** Summary 293
 - Bibliographic Notes 294
 - Homework Problems 294
 - Solutions to Practice Problems 308

4

Processor Architecture 333

- 4.1** The Y86 Instruction Set Architecture 336
 - 4.1.1 Programmer-Visible State 336
 - 4.1.2 Y86 Instructions 337
 - 4.1.3 Instruction Encoding 339
 - 4.1.4 Y86 Exceptions 344
 - 4.1.5 Y86 Programs 345
 - 4.1.6 Some Y86 Instruction Details 350
- 4.2** Logic Design and the Hardware Control Language HCL 352
 - 4.2.1 Logic Gates 353
 - 4.2.2 Combinational Circuits and HCL Boolean Expressions 354
 - 4.2.3 Word-Level Combinational Circuits and HCL Integer Expressions 355
 - 4.2.4 Set Membership 360
 - 4.2.5 Memory and Clocking 361
- 4.3** Sequential Y86 Implementations 364
 - 4.3.1 Organizing Processing into Stages 364
 - 4.3.2 SEQ Hardware Structure 375
 - 4.3.3 SEQ Timing 379
 - 4.3.4 SEQ Stage Implementations 383
- 4.4** General Principles of Pipelining 391
 - 4.4.1 Computational Pipelines 392
 - 4.4.2 A Detailed Look at Pipeline Operation 393
 - 4.4.3 Limitations of Pipelining 394
 - 4.4.4 Pipelining a System with Feedback 398
- 4.5** Pipelined Y86 Implementations 400
 - 4.5.1 SEQ+: Rearranging the Computation Stages 400

- 4.5.2 Inserting Pipeline Registers 401
- 4.5.3 Rearranging and Relabeling Signals 405
- 4.5.4 Next PC Prediction 406
- 4.5.5 Pipeline Hazards 408
- 4.5.6 Avoiding Data Hazards by Stalling 413
- 4.5.7 Avoiding Data Hazards by Forwarding 415
- 4.5.8 Load/Use Data Hazards 418
- 4.5.9 Exception Handling 420
- 4.5.10 PIPE Stage Implementations 423
- 4.5.11 Pipeline Control Logic 431
- 4.5.12 Performance Analysis 444
- 4.5.13 Unfinished Business 446
- 4.6** Summary 449
 - 4.6.1 Y86 Simulators 450
 - Bibliographic Notes 451
 - Homework Problems 451
 - Solutions to Practice Problems 457

5

Optimizing Program Performance 473

- 5.1** Capabilities and Limitations of Optimizing Compilers 476
- 5.2** Expressing Program Performance 480
- 5.3** Program Example 482
- 5.4** Eliminating Loop Inefficiencies 486
- 5.5** Reducing Procedure Calls 490
- 5.6** Eliminating Unneeded Memory References 491
- 5.7** Understanding Modern Processors 496
 - 5.7.1 Overall Operation 497
 - 5.7.2 Functional Unit Performance 500
 - 5.7.3 An Abstract Model of Processor Operation 502
- 5.8** Loop Unrolling 509
- 5.9** Enhancing Parallelism 513
 - 5.9.1 Multiple Accumulators 514
 - 5.9.2 Reassociation Transformation 518
- 5.10** Summary of Results for Optimizing Combining Code 524
- 5.11** Some Limiting Factors 525
 - 5.11.1 Register Spilling 525
 - 5.11.2 Branch Prediction and Misprediction Penalties 526
- 5.12** Understanding Memory Performance 531
 - 5.12.1 Load Performance 531
 - 5.12.2 Store Performance 532

- 5.13 Life in the Real World: Performance Improvement Techniques 539
- 5.14 Identifying and Eliminating Performance Bottlenecks 540
 - 5.14.1 Program Profiling 540
 - 5.14.2 Using a Profiler to Guide Optimization 542
 - 5.14.3 Amdahl's Law 545
- 5.15 Summary 547
 - Bibliographic Notes 548
 - Homework Problems 549
 - Solutions to Practice Problems 552

6

The Memory Hierarchy 559

- 6.1 Storage Technologies 561
 - 6.1.1 Random-Access Memory 561
 - 6.1.2 Disk Storage 570
 - 6.1.3 Solid State Disks 581
 - 6.1.4 Storage Technology Trends 583
- 6.2 Locality 586
 - 6.2.1 Locality of References to Program Data 587
 - 6.2.2 Locality of Instruction Fetches 588
 - 6.2.3 Summary of Locality 589
- 6.3 The Memory Hierarchy 591
 - 6.3.1 Caching in the Memory Hierarchy 592
 - 6.3.2 Summary of Memory Hierarchy Concepts 595
- 6.4 Cache Memories 596
 - 6.4.1 Generic Cache Memory Organization 597
 - 6.4.2 Direct-Mapped Caches 599
 - 6.4.3 Set Associative Caches 606
 - 6.4.4 Fully Associative Caches 608
 - 6.4.5 Issues with Writes 611
 - 6.4.6 Anatomy of a Real Cache Hierarchy 612
 - 6.4.7 Performance Impact of Cache Parameters 614
- 6.5 Writing Cache-friendly Code 615
- 6.6 Putting It Together: The Impact of Caches on Program Performance 620
 - 6.6.1 The Memory Mountain 621
 - 6.6.2 Rearranging Loops to Increase Spatial Locality 625
 - 6.6.3 Exploiting Locality in Your Programs 629
- 6.7 Summary 629
 - Bibliographic Notes 630
 - Homework Problems 631
 - Solutions to Practice Problems 642

Part II Running Programs on a System

7

Linking 653

- 7.1** Compiler Drivers 655
- 7.2** Static Linking 657
- 7.3** Object Files 657
- 7.4** Relocatable Object Files 658
- 7.5** Symbols and Symbol Tables 660
- 7.6** Symbol Resolution 663
 - 7.6.1 How Linkers Resolve Multiply Defined Global Symbols 664
 - 7.6.2 Linking with Static Libraries 667
 - 7.6.3 How Linkers Use Static Libraries to Resolve References 670
- 7.7** Relocation 672
 - 7.7.1 Relocation Entries 672
 - 7.7.2 Relocating Symbol References 673
- 7.8** Executable Object Files 678
- 7.9** Loading Executable Object Files 679
- 7.10** Dynamic Linking with Shared Libraries 681
- 7.11** Loading and Linking Shared Libraries from Applications 683
- 7.12** Position-Independent Code (PIC) 687
- 7.13** Tools for Manipulating Object Files 690
- 7.14** Summary 691
 - Bibliographic Notes 691
 - Homework Problems 692
 - Solutions to Practice Problems 698

8

Exceptional Control Flow 701

- 8.1** Exceptions 703
 - 8.1.1 Exception Handling 704
 - 8.1.2 Classes of Exceptions 706
 - 8.1.3 Exceptions in Linux/IA32 Systems 708
- 8.2** Processes 712
 - 8.2.1 Logical Control Flow 712
 - 8.2.2 Concurrent Flows 713
 - 8.2.3 Private Address Space 714
 - 8.2.4 User and Kernel Modes 714
 - 8.2.5 Context Switches 716

- 8.3** System Call Error Handling 717
- 8.4** Process Control 718
 - 8.4.1 Obtaining Process IDs 719
 - 8.4.2 Creating and Terminating Processes 719
 - 8.4.3 Reaping Child Processes 723
 - 8.4.4 Putting Processes to Sleep 729
 - 8.4.5 Loading and Running Programs 730
 - 8.4.6 Using fork and execve to Run Programs 733
- 8.5** Signals 736
 - 8.5.1 Signal Terminology 738
 - 8.5.2 Sending Signals 739
 - 8.5.3 Receiving Signals 742
 - 8.5.4 Signal Handling Issues 745
 - 8.5.5 Portable Signal Handling 752
 - 8.5.6 Explicitly Blocking and Unblocking Signals 753
 - 8.5.7 Synchronizing Flows to Avoid Nasty Concurrency Bugs 755
- 8.6** Nonlocal Jumps 759
- 8.7** Tools for Manipulating Processes 762
- 8.8** Summary 763
 - Bibliographic Notes 763
 - Homework Problems 764
 - Solutions to Practice Problems 771

9

Virtual Memory 775

- 9.1** Physical and Virtual Addressing 777
- 9.2** Address Spaces 778
- 9.3** VM as a Tool for Caching 779
 - 9.3.1 DRAM Cache Organization 780
 - 9.3.2 Page Tables 780
 - 9.3.3 Page Hits 782
 - 9.3.4 Page Faults 782
 - 9.3.5 Allocating Pages 783
 - 9.3.6 Locality to the Rescue Again 784
- 9.4** VM as a Tool for Memory Management 785
- 9.5** VM as a Tool for Memory Protection 786
- 9.6** Address Translation 787
 - 9.6.1 Integrating Caches and VM 791
 - 9.6.2 Speeding up Address Translation with a TLB 791
 - 9.6.3 Multi-Level Page Tables 792
 - 9.6.4 Putting It Together: End-to-end Address Translation 794
- 9.7** Case Study: The Intel Core i7/Linux Memory System 799

- 9.7.1 Core i7 Address Translation 800
- 9.7.2 Linux Virtual Memory System 803
- 9.8** Memory Mapping 807
 - 9.8.1 Shared Objects Revisited 807
 - 9.8.2 The fork Function Revisited 809
 - 9.8.3 The execve Function Revisited 810
 - 9.8.4 User-level Memory Mapping with the mmap Function 810
- 9.9** Dynamic Memory Allocation 812
 - 9.9.1 The malloc and free Functions 814
 - 9.9.2 Why Dynamic Memory Allocation? 816
 - 9.9.3 Allocator Requirements and Goals 817
 - 9.9.4 Fragmentation 819
 - 9.9.5 Implementation Issues 820
 - 9.9.6 Implicit Free Lists 820
 - 9.9.7 Placing Allocated Blocks 822
 - 9.9.8 Splitting Free Blocks 823
 - 9.9.9 Getting Additional Heap Memory 823
 - 9.9.10 Coalescing Free Blocks 824
 - 9.9.11 Coalescing with Boundary Tags 824
 - 9.9.12 Putting It Together: Implementing a Simple Allocator 827
 - 9.9.13 Explicit Free Lists 835
 - 9.9.14 Segregated Free Lists 836
- 9.10** Garbage Collection 838
 - 9.10.1 Garbage Collector Basics 839
 - 9.10.2 Mark&Sweep Garbage Collectors 840
 - 9.10.3 Conservative Mark&Sweep for C Programs 842
- 9.11** Common Memory-Related Bugs in C Programs 843
 - 9.11.1 Dereferencing Bad Pointers 843
 - 9.11.2 Reading Uninitialized Memory 843
 - 9.11.3 Allowing Stack Buffer Overflows 844
 - 9.11.4 Assuming that Pointers and the Objects They Point to Are the Same Size 844
 - 9.11.5 Making Off-by-One Errors 845
 - 9.11.6 Referencing a Pointer Instead of the Object It Points to 845
 - 9.11.7 Misunderstanding Pointer Arithmetic 846
 - 9.11.8 Referencing Nonexistent Variables 846
 - 9.11.9 Referencing Data in Free Heap Blocks 847
 - 9.11.10 Introducing Memory Leaks 847
- 9.12** Summary 848
 - Bibliographic Notes 848
 - Homework Problems 849
 - Solutions to Practice Problems 853

Part III Interaction and Communication Between Programs

10

System-Level I/O 861

- 10.1** Unix I/O 862
- 10.2** Opening and Closing Files 863
- 10.3** Reading and Writing Files 865
- 10.4** Robust Reading and Writing with the Rio Package 867
 - 10.4.1 Rio Unbuffered Input and Output Functions 867
 - 10.4.2 Rio Buffered Input Functions 868
- 10.5** Reading File Metadata 873
- 10.6** Sharing Files 875
- 10.7** I/O Redirection 877
- 10.8** Standard I/O 879
- 10.9** Putting It Together: Which I/O Functions Should I Use? 880
- 10.10** Summary 881
 - Bibliographic Notes 882
 - Homework Problems 882
 - Solutions to Practice Problems 883

11

Network Programming 885

- 11.1** The Client-Server Programming Model 886
- 11.2** Networks 887
- 11.3** The Global IP Internet 891
 - 11.3.1 IP Addresses 893
 - 11.3.2 Internet Domain Names 895
 - 11.3.3 Internet Connections 899
- 11.4** The Sockets Interface 900
 - 11.4.1 Socket Address Structures 901
 - 11.4.2 The `socket` Function 902
 - 11.4.3 The `connect` Function 903
 - 11.4.4 The `open_clientfd` Function 903
 - 11.4.5 The `bind` Function 904
 - 11.4.6 The `listen` Function 905
 - 11.4.7 The `open_listenfd` Function 905
 - 11.4.8 The `accept` Function 907
 - 11.4.9 Example Echo Client and Server 908

- 11.5** Web Servers 911
 - 11.5.1 Web Basics 911
 - 11.5.2 Web Content 912
 - 11.5.3 HTTP Transactions 914
 - 11.5.4 Serving Dynamic Content 916
- 11.6** Putting It Together: The TINY Web Server 919
- 11.7** Summary 927
 - Bibliographic Notes 928
 - Homework Problems 928
 - Solutions to Practice Problems 929

12

Concurrent Programming 933

- 12.1** Concurrent Programming with Processes 935
 - 12.1.1 A Concurrent Server Based on Processes 936
 - 12.1.2 Pros and Cons of Processes 937
- 12.2** Concurrent Programming with I/O Multiplexing 939
 - 12.2.1 A Concurrent Event-Driven Server Based on I/O Multiplexing 942
 - 12.2.2 Pros and Cons of I/O Multiplexing 946
- 12.3** Concurrent Programming with Threads 947
 - 12.3.1 Thread Execution Model 948
 - 12.3.2 Posix Threads 948
 - 12.3.3 Creating Threads 950
 - 12.3.4 Terminating Threads 950
 - 12.3.5 Reaping Terminated Threads 951
 - 12.3.6 Detaching Threads 951
 - 12.3.7 Initializing Threads 952
 - 12.3.8 A Concurrent Server Based on Threads 952
- 12.4** Shared Variables in Threaded Programs 954
 - 12.4.1 Threads Memory Model 955
 - 12.4.2 Mapping Variables to Memory 956
 - 12.4.3 Shared Variables 956
- 12.5** Synchronizing Threads with Semaphores 957
 - 12.5.1 Progress Graphs 960
 - 12.5.2 Semaphores 963
 - 12.5.3 Using Semaphores for Mutual Exclusion 964
 - 12.5.4 Using Semaphores to Schedule Shared Resources 966
 - 12.5.5 Putting It Together: A Concurrent Server Based on Prethreading 970
- 12.6** Using Threads for Parallelism 974

12.7	Other Concurrency Issues	979
12.7.1	Thread Safety	979
12.7.2	Reentrancy	980
12.7.3	Using Existing Library Functions in Threaded Programs	982
12.7.4	Races	983
12.7.5	Deadlocks	985
12.8	Summary	988
	Bibliographic Notes	989
	Homework Problems	989
	Solutions to Practice Problems	994

A

Error Handling 999

A.1	Error Handling in Unix Systems	1000
A.2	Error-Handling Wrappers	1001

References 1005

Index 1011