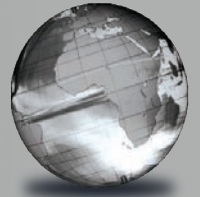


GLOBAL
EDITION



Computer Systems

A Programmer's Perspective

THIRD EDITION

Randal E. Bryant • David R. O'Hallaron

Computer Systems

A Programmer's Perspective

Computer Systems: A Programmer's Perspective, Global Edition

Table of Contents

Front Cover

Dedication

Contents

Preface

About the Authors

Chapter 1: A Tour of Computer Systems

1.1: Information Is Bits + Context

1.2: Programs Are Translated by Other Programs into Different Forms

1.3: It Pays to Understand How Compilation Systems Work

1.4: Processors Read and Interpret Instructions Stored in Memory

1.4.1: Hardware Organization of a System

1.4.2: Running the hello Program

1.5: Caches Matter

1.6: Storage Devices Form a Hierarchy

1.7: The Operating System Manages the Hardware

1.7.1: Processes

1.7.2: Threads

1.7.3: Virtual Memory

1.7.4: Files

1.8: Systems Communicate with Other Systems Using Networks

1.9: Important Themes

1.9.1: Amdahls Law

1.9.2: Concurrency and Parallelism

1.9.3: The Importance of Abstractions in Computer Systems

1.10: Summary

Bibliographic Notes

Solutions to Practice Problems

Part I: Program Structure and Execution

Chapter 2: Representing and Manipulating Information

Table of Contents

2.1: Information Storage

- 2.1.1: Hexadecimal Notation
- 2.1.2: Data Sizes
- 2.1.3: Addressing and Byte Ordering
- 2.1.4: Representing Strings
- 2.1.5: Representing Code
- 2.1.6: Introduction to Boolean Algebra
- 2.1.7: Bit-Level Operations in C
- 2.1.8: Logical Operations in C
- 2.1.9: Shift Operations in C

2.2: Integer Representations

- 2.2.1: Integral Data Types
- 2.2.2: Unsigned Encodings
- 2.2.3: Twos-Complement Encodings
- 2.2.4: Conversions between Signed and Unsigned
- 2.2.5: Signed versus Unsigned in C
- 2.2.6: Expanding the Bit Representation of a Number
- 2.2.7: Truncating Numbers
- 2.2.8: Advice on Signed versus Unsigned

2.3: Integer Arithmetic

- 2.3.1: Unsigned Addition
- 2.3.2: Twos-Complement Addition
- 2.3.3: Twos-Complement Negation
- 2.3.4: Unsigned Multiplication
- 2.3.5: Twos-Complement Multiplication
- 2.3.6: Multiplying by Constant
- 2.3.7: Dividing by Powers of 2
- 2.3.8: Final Thoughts on Integer Arithmetic

2.4: Floating Point

- 2.4.1: Fractional Binary Numbers
- 2.4.2: IEEE Floating-Point Representation
- 2.4.3: Example Numbers
- 2.4.4: Rounding
- 2.4.5: Floating-Point Operations
- 2.4.6: Floating Point in C

2.5: Summary

- Bibliographic Notes
- Homework Problems
- Solutions to Practice Problems

Chapter 3: Machine-Level Representation of Program

- 3.1: A Historical Perspective
- 3.2: Program Encodings

Table of Contents

3.2.1: Machine-Level Code	
3.2.2: Code Examples	
3.2.3: Notes on Formatting	
3.3: Data Formats	
3.4: Accessing Information	
3.4.1: Operand Specifiers	
3.4.2: Data Movement Instructions	
3.4.3: Data Movement Example	
3.4.4: Pushing and Popping Stack Data	
3.5: Arithmetic and Logical Operations	
3.5.1: Load Effective Address	
3.5.2: Unary and Binary Operations	
3.5.3: Shift Operations	
3.5.4: Discussion	
3.5.5: Special Arithmetic Operations	
3.6: Control	
3.6.1: Condition Codes	
3.6.2: Accessing the Condition Codes	
3.6.3: Jump Instructions	
3.6.4: Jump Instruction Encodings	
3.6.5: Implementing Conditional Branches with Conditional Control	
3.6.6: Implementing Conditional Branches with Conditional Moves	
3.6.7: Loop	
3.6.8: Switch Statements	
3.7: Procedures	
3.7.1: The Run-Time Stack	
3.7.2: Control Transfer	
3.7.3: Data Transfer	
3.7.4: Local Storage on the Stack	
3.7.5: Local Storage in Registers	
3.7.6: Recursive Procedures	
3.8: Array Allocation and Access	
3.8.1: Basic Principles	
3.8.2: Pointer Arithmetic	
3.8.3: Nested Arrays	
3.8.4: Fixed-Size Arrays	
3.8.5: Variable-Size Arrays	
3.9: Heterogeneous Data Structure	
3.9.1: Structures	
3.9.2: Unions	
3.9.3: Data Alignment	
3.10: Combining Control and Data in Machine-Level Programs	

Table of Contents

- 3.10.1: Understanding Pointers
- 3.10.2: Life in the RealWorld: Using the GDB Debugger
- 3.10.3: Out-of-Bounds Memory References and Buffer Overflow
- 3.10.4: Thwarting Buffer Overflow Attacks
- 3.10.5: Supporting Variable-Size Stack Frames

3.11: Floating-Point Code

- 3.11.1: Floating-Point Movement and Conversion Operations
- 3.11.2: Floating-Point Code in Procedures
- 3.11.3: Floating-Point Arithmetic Operations
- 3.11.4: Defining and Using Floating-Point Constants
- 3.11.5: Using Bitwise Operations in Floating-Point Code
- 3.11.6: Floating-Point Comparison Operations
- 3.11.7: Observations about Floating-Point Code

3.12: Summary

- Bibliographic Notes
- Homework Problems
- Solutions to Practice Problems

Chapter 4: Processor Architecture

4.1: The Y86-64 Instruction Set Architecture

- 4.1.1: Programmer-Visible State
- 4.1.2: Y86-64 Instructions
- 4.1.3: Instruction Encoding
- 4.1.4: Y86-64 Exceptions
- 4.1.5: Y86-64 Programs
- 4.1.6: Some Y86-64 Instruction Details

4.2: Logic Design and the Hardware Control Language HCL

- 4.2.1: Logic Gates
- 4.2.2: Combinational Circuits and HCL Boolean Expressions
- 4.2.3: Word-Level Combinational Circuits and HCL Integer Expressions
- 4.2.4: Set Membership
- 4.2.5: Memory and Clocking

4.3: Sequential Y86-64 Implementations

- 4.3.1: Organizing Processing into Stages
- 4.3.2: SEQ Hardware Structure
- 4.3.3: SEQ Timing
- 4.3.4: SEQ Stage Implementations

4.4: General Principles of Pipelining

- 4.4.1: Computational Pipelines
- 4.4.2: A Detailed Look at Pipeline Operation
- 4.4.3: Limitations of Pipelining
- 4.4.4: Pipelining a System with Feedback

4.5: Pipelined Y86-64 Implementations

Table of Contents

- 4.5.1: SEQ+: Rearranging the Computation Stages
- 4.5.2: Inserting Pipeline Registers
- 4.5.3: Rearranging and Relabeling Signals
- 4.5.4: Next PC Prediction
- 4.5.5: Pipeline Hazards
- 4.5.6: Exception Handling
- 4.5.7: PIPE Stage Implementations
- 4.5.8: Pipeline Control Logic
- 4.5.9: Performance Analysis
- 4.5.10: Unfinished Business
- 4.6: Summary
 - 4.6.1: Y86-64 Simulators
 - Bibliographic Notes
 - Homework Problems
 - Solutions to Practice Problems

Chapter 5: Optimizing Program Performance

- 5.1: Capabilities and Limitations of Optimizing Compilers
- 5.2: Expressing Program Performance
- 5.3: Program Example
- 5.4: Eliminating Loop Inefficiencies
- 5.5: Reducing Procedure Calls
- 5.6: Eliminating Unneeded Memory References
- 5.7: Understanding Modern Processors
 - 5.7.1: Overall Operation
 - 5.7.2: Functional Unit Performance
 - 5.7.3: An Abstract Model of Processor Operation
- 5.8: Loop Unrolling
- 5.9: Enhancing Parallelism
 - 5.9.1: Multiple Accumulators
 - 5.9.2: Reassociation Transformation
- 5.10: Summary of Results for Optimizing Combining Code
- 5.11: Some Limiting Factors
 - 5.11.1: Register Spilling
 - 5.11.2: Branch Prediction and Misprediction Penalties
- 5.12: Understanding Memory Performance
 - 5.12.1: Load Performance
 - 5.12.2: Store Performance
- 5.13: Life in the Real World: Performance Improvement Techniques
- 5.14: Identifying and Eliminating Performance Bottlenecks
 - 5.14.1: Program Profiling

Table of Contents

5.14.2: Using a Profiler to Guide Optimization

5.15: Summary

Bibliographic Notes

Homework Problems

Solutions to Practice Problems

Chapter 6: The Memory Hierarchy

6.1: Storage Technologie

6.1.1: Random Access Memory

6.1.2: Disk Storage

6.1.3: Solid State Disks

6.1.4: Storage Technology Trends

6.2: Locality

6.2.1: Locality of References to Program Data

6.2.2: Locality of Instruction Fetches

6.2.3: Summary of Locality

6.3: The Memory Hierarchy

6.3.1: Caching in the Memory Hierarchy

6.3.2: Summary of Memory Hierarchy Concepts

6.4: Cache Memories

6.4.1: Generic Cache Memory Organization

6.4.2: Direct-Mapped Caches

6.4.3: Set Associative Caches

6.4.4: Fully Associative Caches

6.4.5: Issues with Writes

6.4.6: Anatomy of a Real Cache Hierarchy

6.4.7: Performance Impact of Cache Parameters

6.5: Writing Cache-Friendly Code

6.6: Putting It Together: The Impact of Caches on Program Performance

6.6.1: The Memory Mountain

6.6.2: Rearranging Loops to Increase Spatial Locality

6.6.3: Exploiting Locality in Your Programs

6.7: Summary

Bibliographic Notes

Homework Problems

Solutions to Practice Problems

Part II: Running Programs on a System

Chapter 7: Linking

7.1: Compiler Drivers

7.2: Static Linking

7.3: Object Files

Table of Contents

- 7.4: Relocatable Object Files
- 7.5: Symbols and Symbol Tables
- 7.6: Symbol Resolution
 - 7.6.1: How Linkers Resolve Duplicate Symbol Names
 - 7.6.2: Linking with Static Libraries
 - 7.6.3: How Linkers Use Static Libraries to Resolve References
- 7.7: Relocation
 - 7.7.1: Relocation Entries
 - 7.7.2: Relocating Symbol References
- 7.8: Executable Object Files
- 7.9: Loading Executable Object Files
- 7.10: Dynamic Linking with Shared Libraries
- 7.11: Loading and Linking Shared Libraries from Applications
- 7.12: Position-Independent Code (PIC)
- 7.13: Library Interpositioning
 - 7.13.1: Compile-Time Interpositioning
 - 7.13.2: Link-Time Interpositioning
 - 7.13.3: Run-Time Interpositioning
- 7.14: Tools for Manipulating Object Files
- 7.15: Summary
 - Bibliographic Notes
 - Homework Problems
 - Solutions to Practice Problems

Chapter 8: Exceptional Control Flow

- 8.1: Exceptions
 - 8.1.1: Exception Handling
 - 8.1.2: Classes of Exceptions
 - 8.1.3: Exceptions in Linux/x86-64 Systems
- 8.2: Processes
 - 8.2.1: Logical Control Flow
 - 8.2.2: Concurrent Flows
 - 8.2.3: Private Address Space
 - 8.2.4: User and Kernel Modes
 - 8.2.5: Context Switches
- 8.3: System Call Error Handling
- 8.4: Process Control
 - 8.4.1: Obtaining Process IDs
 - 8.4.2: Creating and Terminating Processes
 - 8.4.3: Reaping Child Processes
 - 8.4.4: Putting Processes to Sleep

Table of Contents

- 8.4.5: Loading and Running Programs
- 8.4.6: Using fork and execve to Run Programs

8.5: Signals

- 8.5.1: Signal Terminology
- 8.5.2: Sending Signals
- 8.5.3: Receiving Signals
- 8.5.4: Blocking and Unblocking Signals
- 8.5.5: Writing Signal Handlers
- 8.5.6: Synchronizing Flows to Avoid Nasty Concurrency Bugs
- 8.5.7: ExplicitlyWaiting for Signals

8.6: Nonlocal Jumps

8.7: Tools for Manipulating Processes

8.8: Summary

- Bibliographic Notes
- Homework Problems
- Solutions to Practice Problems

Chapter 9: Virtual Memory

9.1: Physical and Virtual Addressing

9.2: Address Spaces

9.3: VM as a Tool for Caching

- 9.3.1: DRAM Cache Organization
- 9.3.2: Page Tables
- 9.3.3: Page Hits
- 9.3.4: Page Faults
- 9.3.5: Allocating Pages
- 9.3.6: Locality to the Rescue Again

9.4: VM as a Tool for Memory Management

9.5: VM as a Tool for Memory Protection

9.6: Address Translation

- 9.6.1: Integrating Caches and VM
- 9.6.2: Speeding Up Address Translation with a TLB
- 9.6.3: Multi-Level Page Tables
- 9.6.4: Putting It Together: End-to-End Address Translation

9.7: Case Study: The Intel Core i7/Linux Memory System

- 9.7.1: Core i7 Address Translation
- 9.7.2: Linux Virtual Memory System

9.8: Memory Mapping

- 9.8.1: Shared Objects Revisited
- 9.8.2: The fork Function Revisited
- 9.8.3: The execve Function Revisited
- 9.8.4: User-Level Memory Mapping with the mmap Function

Table of Contents

9.9: Dynamic Memory Allocation

- 9.9.1: The malloc and free Functions
- 9.9.2: Why Dynamic Memory Allocation?
- 9.9.3: Allocator Requirements and Goals
- 9.9.4: Fragmentation
- 9.9.5: Implementation Issues
- 9.9.6: Implicit Free Lists
- 9.9.7: Placing Allocated Blocks
- 9.9.8: Splitting Free Blocks
- 9.9.9: Getting Additional Heap Memory
- 9.9.10: Coalescing Free Blocks
- 9.9.11: Coalescing with Boundary Tags
- 9.9.12: Putting It Together: Implementing a Simple Allocator
- 9.9.13: Explicit Free Lists
- 9.9.14: Segregated Free Lists

9.10: Garbage Collection

- 9.10.1: Garbage Collector Basics
- 9.10.2: Mark&Sweep Garbage Collectors
- 9.10.3: Conservative Mark&Sweep for C Programs

9.11: Common Memory-Related Bugs in C Programs

- 9.11.1: Dereferencing Bad Pointers
- 9.11.2: Reading Uninitialized Memory
- 9.11.3: Allowing Stack Buffer Overflows
- 9.11.4: Assuming That Pointers and the Objects They Point to Are the Same Size
- 9.11.5: Making Off-by-One Errors
- 9.11.6: Referencing a Pointer Instead of the Object It Points To
- 9.11.7: Misunderstanding Pointer Arithmetic
- 9.11.8: Referencing Nonexistent Variables
- 9.11.9: Referencing Data in Free Heap Blocks
- 9.11.10: Introducing Memory Leaks

9.12: Summary

- Bibliographic Notes
- Homework Problems
- Solutions to Practice Problems

Part III: Interaction and Communication between Programs

Chapter 10: System-Level I/O

- 10.1: Unix I/O
- 10.2: Files
- 10.3: Opening and Closing Files
- 10.4: Reading and Writing Files
- 10.5: Robust Reading and Writing with the Rio Package

Table of Contents

10.5.1: Rio Unbuffered Input and Output Functions

10.5.2: Rio Buffered Input Functions

10.6: Reading File Metadata

10.7: Reading Directory Contents

10.8: Sharing Files

10.9: I/O Redirection

10.10: Standard I/O

10.11: Putting It Together: Which I/O Functions Should I Use?

10.12: Summary

Bibliographic Notes

Homework Problems

Solutions to Practice Problems

Chapter 11: Network Programming

11.1: The Client-Server Programming Model

11.2: Networks

11.3: The Global IP Internet

11.3.1: IP Addresses

11.3.2: Internet Domain Names

11.3.3: Internet Connections

11.4: The Sockets Interface

11.4.1: Socket Address Structures

11.4.2: The socket Function

11.4.3: The connect Function

11.4.4: The bind Function

11.4.5: The listen Function

11.4.6: The accept Function

11.4.7: Host and Service Conversion

11.4.8: Helper Functions for the Sockets Interface

11.4.9: Example Echo Client and Server

11.5: Web Servers

11.5.1: Web Basics

11.5.2: Web Content

11.5.3: HTTP Transactions

11.5.4: Serving Dynamic Content

11.6: Putting It Together: The Tiny Web Server

11.7: Summary

Bibliographic Notes

Homework Problems

Solutions to Practice Problems

Chapter 12: Concurrent Programming

Table of Contents

12.1: Concurrent Programming with Processes

12.1.1: A Concurrent Server Based on Processes

12.1.2: Pros and Cons of Processes

12.2: Concurrent Programming with I/O Multiplexing

12.2.1: A Concurrent Event-Driven Server Based on I/O Multiplexing

12.2.2: Pros and Cons of I/O Multiplexing

12.3: Concurrent Programming with Threads

12.3.1: Thread Execution Model

12.3.2: Posix Threads

12.3.3: Creating Threads

12.3.4: Terminating Threads

12.3.5: Reaping Terminated Threads

12.3.6: Detaching Threads

12.3.7: Initializing Threads

12.3.8: A Concurrent Server Based on Threads

12.4: Shared Variables in Threaded Programs

12.4.1: Threads Memory Model

12.4.2: Mapping Variables to Memory

12.4.3: Shared Variables

12.5: Synchronizing Threads with Semaphores

12.5.1: Progress Graphs

12.5.2: Semaphores

12.5.3: Using Semaphores for Mutual Exclusion

12.5.4: Using Semaphores to Schedule Shared Resources

12.5.5: Putting It Together: A Concurrent Server Based on Prethreading

12.6: Using Threads for Parallelism

12.7: Other Concurrency Issues

12.7.1: Thread Safety

12.7.2: Reentrancy

12.7.3: Using Existing Library Functions in Threaded Programs

12.7.4: Races

12.7.5: Deadlocks

12.8: Summary

Bibliographic Notes

Homework Problems

Solutions to Practice Problems

Appendix A: Error Handling

A.1: Error Handling in Unix Systems

A.2: Error-Handling Wrappers

References

Table of Contents

Index

Back Cover