

---

# Contents

- Chapter 1** Origins of Mac OS X
  - 1.1 Apple's Quest for *the* Operating System
    - 1.1.1 Star Trek
    - 1.1.2 Raptor
    - 1.1.3 NuKernel
    - 1.1.4 TalOS
    - 1.1.5 Copland
    - 1.1.6 Gershwin
    - 1.1.7 BeOS
    - 1.1.8 Plan A
  - 1.2 The NeXT Chapter
    - 1.2.1 NEXTSTEP
    - 1.2.2 OPENSTEP
  - 1.3 The Mach Factor
    - 1.3.1 Rochester's Intelligent Gateway
    - 1.3.2 Accent
    - 1.3.3 Mach
    - 1.3.4 MkLinux
    - 1.3.5 Musical Names
  - 1.4 Strategies
    - 1.4.1 Mac OS 8 and 9
    - 1.4.2 Rhapsody
      - 1.4.2.1 Blue Box
      - 1.4.2.2 Yellow Box
  - 1.5 Toward Mac OS X

- 1.5.1 Mac OS X Server 1.x
- 1.5.2 Mac OS X Developer Previews
  - 1.5.2.1 DP1
  - 1.5.2.2 DP2
  - 1.5.2.3 DP3
  - 1.5.2.4 DP4
- 1.5.3 Mac OS X Public Beta
- 1.5.4 Mac OS X 10.x
  - 1.5.4.1 Mac OS X 10.0
  - 1.5.4.2 Mac OS X 10.1
  - 1.5.4.3 Mac OS X 10.2
  - 1.5.4.4 Mac OS X 10.3
  - 1.5.4.5 Mac OS X 10.4

## **Chapter 2** An Overview of Mac OS X

- 2.1 Firmware
- 2.2 Bootloader
- 2.3 Darwin
  - 2.3.1 Darwin Packages
  - 2.3.2 The Benefits of Darwin
  - 2.3.3 Darwin and Mac OS X
- 2.4 The xnu Kernel
  - 2.4.1 Mach
  - 2.4.2 BSD
  - 2.4.3 The I/O Kit
  - 2.4.4 The libkern Library
  - 2.4.5 The libsa Library
  - 2.4.6 The Platform Expert
  - 2.4.7 Kernel Extensions
- 2.5 A User-Space View of the File System
  - 2.5.1 File System Domains
    - 2.5.1.1 The User Domain
    - 2.5.1.2 The Local Domain
    - 2.5.1.3 The Network Domain
    - 2.5.1.4 The System Domain
  - 2.5.2 The /System/Library/ Directory

- 2.6 The Runtime Architecture
  - 2.6.1 Mach-O Files
  - 2.6.2 Fat Binaries
  - 2.6.3 Linking
    - 2.6.3.1 Multiple Binding Styles
    - 2.6.3.2 Two-Level Namespaces
    - 2.6.3.3 Weakly Linked Symbols
    - 2.6.3.4 dyld Interposing
- 2.7 The C Library
- 2.8 Bundles and Frameworks
  - 2.8.1 Bundles
  - 2.8.2 Property List Files
  - 2.8.3 Frameworks
  - 2.8.4 Prebinding
- 2.9 Core Services
- 2.10 Application Services
  - 2.10.1 Graphics and Multimedia Services
    - 2.10.1.1 Quartz
    - 2.10.1.2 QuickDraw 2D
    - 2.10.1.3 OpenGL
    - 2.10.1.4 Core Image and Core Video
    - 2.10.1.5 QuickTime
    - 2.10.1.6 Core Audio
  - 2.10.2 Other Application Services
- 2.11 Application Environments
  - 2.11.1 BSD
  - 2.11.2 The X Window System
  - 2.11.3 Carbon
    - 2.11.3.1 Support for CFM Binaries
    - 2.11.3.2 Carbon APIs
  - 2.11.4 Cocoa
    - 2.11.4.1 Nib Files
    - 2.11.4.2 Core Data
  - 2.11.5 WebObjects
  - 2.11.6 Java
  - 2.11.7 QuickTime
  - 2.11.8 Classic
  - 2.11.9 Rosetta
- 2.12 User Interface
  - 2.12.1 Visual Effects

- 2.12.2 Resolution-Independent User Interface
- 2.12.3 Productivity Features
  - 2.12.3.1 Fast User Switching
  - 2.12.3.2 Dashboard
  - 2.12.3.3 Spotlight
- 2.12.4 Universal Access Support
- 2.13 Programming
  - 2.13.1 Xcode
  - 2.13.2 Compilers and Libraries
  - 2.13.3 Interpreters
    - 2.13.3.1 AppleScript
    - 2.13.3.2 Automator
    - 2.13.3.3 Command-Line Support
  - 2.13.4 Tools
    - 2.13.4.1 Debugging and Analysis Tools
    - 2.13.4.2 Computer Hardware Understanding Development Tools
    - 2.13.4.3 Visual Tools
- 2.14 Security
  - 2.14.1 Kernel-Space Security
  - 2.14.2 User-Space Security
    - 2.14.2.1 CDSA Plug-ins
    - 2.14.2.2 CSSM API
    - 2.14.2.3 Mac OS X Security APIs
    - 2.14.2.4 Security Server and Security Agent
    - 2.14.2.5 Using Authorization Services
    - 2.14.2.6 Miscellaneous Security-Related Features
  - 2.14.3 System Administration
    - 2.14.3.1 Interacting with the Security Framework
    - 2.14.3.2 Interacting with Directory Services
    - 2.14.3.3 Managing System Configuration
  - 2.14.4 The Auditing System
- 2.15 Mac OS X Server
  - 2.15.1 Xgrid
    - 2.15.1.1 Xgrid Architecture
    - 2.15.1.2 Xgrid Software
  - 2.15.2 Xsan
    - 2.15.2.1 Storage in Xsan
    - 2.15.2.2 Metadata Controllers
    - 2.15.2.3 Client Systems

- 2.15.2.4 Communication Infrastructure
- 2.16 Networking

## **Chapter 3** Inside an Apple

- 3.1 The Power Mac G5
  - 3.1.1 The U3H System Controller
  - 3.1.2 The K2 I/O Device Controller
  - 3.1.3 PCI-X and PCI Express
    - 3.1.3.1 PCI-X
    - 3.1.3.2 PCI Express
  - 3.1.4 HyperTransport
  - 3.1.5 Elastic I/O Interconnect
- 3.2 The G5: Lineage and Roadmap
  - 3.2.1 Fundamental Aspects of the G5
    - 3.2.1.1 64-bit Processor
    - 3.2.1.2 Superscalar
    - 3.2.1.3 Speculative Execution
    - 3.2.1.4 Out-of-Order Execution
  - 3.2.2 New POWER Generations
  - 3.2.3 The PowerPC 970, 970FX, and 970MP
  - 3.2.4 The Intel Core Duo
- 3.3 The PowerPC 970FX
  - 3.3.1 At a Glance
  - 3.3.2 Caches
    - 3.3.2.1 L1 and L2 Caches
    - 3.3.2.2 Cache Properties
      - 3.3.2.2 Associativity
      - 3.3.2.2 Store Policy
      - 3.3.2.2 MERSI
  - 3.3.3 Memory Management Unit (MMU)
    - 3.3.3.1 SLB and TLB
    - 3.3.3.2 Address Translation
    - 3.3.3.3 Caching the Caches: ERATs
    - 3.3.3.4 Large Pages
    - 3.3.3.5 No Support for Block Address Translation Mechanism
  - 3.3.4 Miscellaneous Internal Buffers and Queues
  - 3.3.5 Prefetching
  - 3.3.6 Registers
    - 3.3.6.1 UISA and VEA Registers

- 3.3.6.2 OEA Registers
- 3.3.7 Rename Registers
- 3.3.8 Instruction Set
  - 3.3.8.1 Fixed-Point Instructions
  - 3.3.8.2 Floating-Point Instructions
  - 3.3.8.3 Vector Instructions
  - 3.3.8.4 Control-Flow Instructions
  - 3.3.8.5 Miscellaneous Instructions
- 3.3.9 The 970FX Core
  - 3.3.9.1 Instruction Pipeline
    - 3–14 IFAR, ICA
    - 3–14 D0
    - 3–14 D1, D2, D3
    - 3–15 Accounting for 215 In-Flight Instructions
  - 3.3.9.2 Branch Prediction
  - 3.3.9.3 Summary
- 3.3.10 AltiVec
  - 3.3.10.1 Vector Computing
  - 3.3.10.2 The 970FX AltiVec Implementation
  - 3.3.10.3 AltiVec Instructions
- 3.3.11 Power Management
  - 3.3.11.1 PowerTune
  - 3.3.11.2 Power Mac G5 Thermal and Power Management
- 3.3.12 64-bit Architecture
  - 3.3.12.1 64-bit Features
  - 3.3.12.2 The 970FX as a 32-bit Processor
- 3.3.13 Softpatch Facility
- 3.4 Software Conventions
  - 3.4.1 Byte Ordering
  - 3.4.2 Register Usage
    - 3.4.2.1 Indirect Calls
    - 3.4.2.2 Direct Calls
  - 3.4.3 Stack Usage
    - 3.4.3.1 Stack Usage Examples
    - 3.4.3.2 Printing Stack Frames
  - 3.4.4 Function Parameters and Return Values
    - 3.4.4.1 Passing Parameters
    - 3.4.4.2 Returning Values
- 3.5 Examples
  - 3.5.1 A Recursive Factorial Function

- 3.5.2 An Atomic Compare-and-Store Function
- 3.5.3 Function Rerouting
  - 3.5.3.1 Instruction Patching
  - 3.5.3.2 Constructing Branch Instructions
- 3.5.4 Cycle-Accurate Simulation of the 970FX

## **Chapter 4** The Firmware and the Bootloader

- 4.1 Introduction
  - 4.1.1 Varieties of Firmware
  - 4.1.2 Preferential Storage
- 4.2 A Whole New World
  - 4.2.1 “New” Is Good News
  - 4.2.2 Modern Boot ROM (PowerPC)
- 4.3 Power-On Reset
- 4.4 Open Firmware
  - 4.4.1 Interacting with Open Firmware
    - 4.4.1.1 Forth Shell
    - 4.4.1.2 TELNET
    - 4.4.1.3 TFTP
    - 4.4.1.4 Serial Download
  - 4.4.2 Open Firmware Emulators
- 4.5 Forth
  - 4.5.1 Cells
  - 4.5.2 Stacks
  - 4.5.3 Words
  - 4.5.4 Dictionary
    - 4.5.4.1 A Sampling of Built-in Words
      - 4.5.4.1 Stacks
      - 4.5.4.1 Memory
      - 4.5.4.1 Operators
      - 4.5.4.1 Console I/O
      - 4.5.4.1 Control Flow
    - 4.5.4.2 Searching the Dictionary
  - 4.5.5 Debugging
- 4.6 The Device Tree
  - 4.6.1 Properties
  - 4.6.2 Methods
  - 4.6.3 Data

- 4.7 Open Firmware Interfaces
  - 4.7.1 The User Interface
  - 4.7.2 The Client Interface
  - 4.7.3 The Device Interface
- 4.8 Programming Examples
  - 4.8.1 Dumping NVRAM Contents
  - 4.8.2 Determining Screen Dimensions
  - 4.8.3 Working with Colors
  - 4.8.4 Drawing a Color-Filled Rectangle
  - 4.8.5 Creating an Animated Solution to the “Towers of Hanoi” Problem
  - 4.8.6 Fabricating and Using a Mouse Pointer
  - 4.8.7 Stealing a Font
  - 4.8.8 Implementing a Clock
  - 4.8.9 Drawing Images
  - 4.8.10 Creating Windows
- 4.9 Firmware Boot Sequence
  - 4.9.1 The Script
  - 4.9.2 Snag Keys
- 4.10 BootX
  - 4.10.1 File Format
  - 4.10.2 Structure
  - 4.10.3 Operation
- 4.11 Alternate Booting Scenarios
  - 4.11.1 Booting an Alternate Kernel
    - 4.11.1.1 NVRAM Caveats
  - 4.11.2 Booting from a Software RAID Device
  - 4.11.3 Booting over a Network
- 4.12 Firmware Security
  - 4.12.1 Managing Firmware Security
  - 4.12.2 Recovering the Open Firmware Password
- 4.13 Launching the Kernel
- 4.14 The BootCache Optimization
- 4.15 Boot-Time Kernel Arguments
- 4.16 The Extensible Firmware Interface
  - 4.16.1 Legacy Pains
  - 4.16.2 A New Beginning
  - 4.16.3 EFI
    - 4.16.3.1 EFI Services
    - 4.16.3.1 Boot Services



- 4.16.3.2 EFI Drivers
- 4.16.4 A Sampling of EFI
  - 4.16.4.1 EFI NVRAM
  - 4.16.4.2 The Boot Manager
  - 4.16.4.3 The EFI Shell
  - 4.16.4.4 The GUID-Based Partitioning Scheme
  - 4.16.4.5 Universal Graphics Adapter
  - 4.16.4.6 EFI Byte Code
  - 4.16.4.7 Binary Format
- 4.16.5 The Benefits of EFI

## **Chapter 5** Kernel and User-Level Startup

- 5.1 Arranging for the Kernel to Execute
  - 5.1.1 Exceptions and Exception Vectors
  - 5.1.2 Kernel Symbols
  - 5.1.3 Run Kernel Run
- 5.2 Low-Level Processor Initialization
  - 5.2.1 Per-Processor Data
  - 5.2.2 Reset Types
  - 5.2.3 Processor Types
  - 5.2.4 Memory Patching
  - 5.2.5 Processor-Specific Initialization
  - 5.2.6 Other Early Initialization
- 5.3 High-Level Processor Initialization
  - 5.3.1 Before Virtual Memory
  - 5.3.2 Low-Level Virtual Memory Initialization
    - 5.3.2.1 Sizing Memory
    - 5.3.2.2 Pmap Initialization
    - 5.3.2.3 Starting Address-Translation
  - 5.3.3 After Virtual Memory
    - 5.3.3.1 Console Initialization
    - 5.3.3.2 Preparing for the Bootstrapping of Kernel Subsystems
- 5.4 Mach Subsystem Initialization
  - 5.4.1 Scheduler Initialization
  - 5.4.2 High-Level Virtual Memory Subsystem Initialization
  - 5.4.3 IPC Initialization
  - 5.4.4 Finishing VM and IPC Initialization
  - 5.4.5 Initializing Miscellaneous Subsystems
  - 5.4.6 Tasks and Threads

- 5.4.7 Launching the Kernel Bootstrap Thread
- 5.5 The First Thread
- 5.6 I/O Kit Initialization
- 5.7 BSD Initialization
  - 5.7.1 Miscellaneous BSD Initialization (Part 1)
  - 5.7.2 File System Initialization
  - 5.7.3 Miscellaneous BSD Initialization (Part 2)
  - 5.7.4 Networking Subsystem Initialization
  - 5.7.5 Miscellaneous BSD Initialization (Part 3)
  - 5.7.6 Mounting the Root File System
  - 5.7.7 Creating Process 1
  - 5.7.8 Shared Memory Regions
- 5.8 Launching the First User-Space Program
- 5.9 Slave Processors
- 5.10 User-Level Startup
  - 5.10.1 launchd
    - 5.10.1.1 Daemon Configuration and Management
    - 5.10.1.2 Daemon Creation
    - 5.10.1.3 launchd Operation
  - 5.10.2 Multiuser Startup
    - 5.10.2.1 User Login
    - 5.10.2.2 User Logout, System Restart, and System Shutdown
  - 5.10.3 Single-User Startup
  - 5.10.4 Installation Startup

## **Chapter 6** The xnu Kernel

- 6.1 xnu Source
- 6.2 Mach
  - 6.2.1 Kernel Fundamentals
    - 6.2.1.1 Tasks and Threads
    - 6.2.1.2 Ports
    - 6.2.1.3 Messages
    - 6.2.1.4 Virtual Memory and Memory Objects
  - 6.2.2 Exception Handling
- 6.3 A Flavor of the Mach APIs
  - 6.3.1 Displaying Host Information
  - 6.3.2 Accessing the Kernel's Clock Services
  - 6.3.3 Using a Clock Service to Ring an Alarm

- 6.3.4 Displaying Host Statistics
- 6.4 Entering the Kernel
  - 6.4.1 Types of Control Transfer
    - 6.4.1.1 External Hardware Interrupts
    - 6.4.1.2 Processor Traps
    - 6.4.1.3 Software Traps
    - 6.4.1.4 System Calls
  - 6.4.2 Implementing System Entry Mechanisms
    - 6.4.2.1 Exceptions and Exception Vectors
    - 6.4.2.2 Exception-Handling Registers
    - 6.4.2.3 System Linkage Instructions
      - 6.4.2.3 System Call
      - 6.4.2.3 Return from Interrupt
    - 6.4.2.4 Machine-Dependent Thread State
    - 6.4.2.5 Exception Save Areas
- 6.5 Exception Processing
  - 6.5.1 Hardware Interrupts
  - 6.5.2 Miscellaneous Traps
  - 6.5.3 System Calls
- 6.6 System Call Processing
- 6.7 System Call Categories
  - 6.7.1 BSD System Calls
    - 6.7.1.1 Data Structures
    - 6.7.1.2 Argument Munging
    - 6.7.1.3 Kernel Processing of BSD System Calls
    - 6.7.1.4 User Processing of BSD System Calls
  - 6.7.2 Mach Traps
  - 6.7.3 I/O Kit Traps
  - 6.7.4 PowerPC-Only System Calls
  - 6.7.5 Ultra-Fast Traps
    - 6.7.5.1 Fast Traps
    - 6.7.5.2 Blue Box Calls
  - 6.7.6 The Commpage
- 6.8 Kernel Support for Debugging, Diagnostics, and Tracing
  - 6.8.1 GDB (Network-Based or FireWire-Based Debugging)
  - 6.8.2 KDB (Serial-Line-Based Debugging)
  - 6.8.3 CHUD Support
    - 6.8.3.1 Task-Related Functions
    - 6.8.3.2 Thread-Related Functions
    - 6.8.3.3 Memory-Related Functions

- 6.8.3.4 CPU-Related Functions
- 6.8.3.5 Callback-Related Functions
- 6.8.4 Kernel Profiling (`kgmon` and `gprof`)
  - 6.8.4.1 Per-Process Profiling (`profil(2)`)
  - 6.8.4.2 Mach Task and Thread Sampling
- 6.8.5 Per-Process Kernel Tracing (`kttrace(2)` and `kdump`)
- 6.8.6 Auditing Support
- 6.8.7 Fine-Grained Kernel Event Tracing (`kdebug`)
  - 6.8.7.1 `kdebug` Tracing
  - 6.8.7.2 `kdebug` Entropy Collection
- 6.8.8 Low-Level Diagnostics and Debugging Interfaces
  - 6.8.8.1 Firmware Call Interface
  - 6.8.8.2 Diagnostics System Call Interface
- 6.8.9 Low-Level Kernel Tracing
  - 6.8.9.1 Low-Memory Global Data Structures
  - 6.8.9.2 Low Tracing
- 6.9 Virtual Machine Monitor
  - 6.9.1 Features
  - 6.9.2 Using the VMM Facility
  - 6.9.3 Example: Running Code in a Virtual Machine
- 6.10 Compiling the Kernel
  - 6.10.1 Retrieving Prerequisite Packages
  - 6.10.2 Compiling Prerequisite Packages
  - 6.10.3 Compiling the `xnu` Package
  - 6.10.4 DarwinBuild

## **Chapter 7** Processes

- 7.1 Processes: From Early UNIX to Mac OS X
  - 7.1.1 Mac OS X Process Limits
  - 7.1.2 Mac OS X Execution Flavors
- 7.2 Mach Abstractions, Data Structures, and APIs
  - 7.2.1 Summary of Relationships
  - 7.2.2 Processor Sets
    - 7.2.2.1 Representation
    - 7.2.2.2 The Processor Set API
  - 7.2.3 Processors
    - 7.2.3.1 Interconnections
    - 7.2.3.2 The Processor API
    - 7.2.3.3 Displaying Processor Information

- 7.2.3.4 Stopping and Starting a Processor in a Multiprocessor System
- 7.2.4 Tasks and the Task API
- 7.2.5 Threads
  - 7.2.5.1 The Thread API
  - 7.2.5.2 Kernel Threads
- 7.2.6 Thread-Related Abstractions
  - 7.2.6.1 Remote Procedure Call
  - 7.2.6.2 Activation and Shuttle
  - 7.2.6.3 Thread Migration
  - 7.2.6.4 Continuations
- 7.3 Many Threads of a New System
  - 7.3.1 Mach Tasks and Threads
    - 7.3.1.1 Creating a Mach Task
    - 7.3.1.2 Creating a Mach Thread in an Existing Task
    - 7.3.1.3 Displaying Task and Thread Details
  - 7.3.2 BSD Processes
    - 7.3.2.1 The `fork()` System Call: User-Space Implementation
    - 7.3.2.2 The `fork()` System Call: Kernel Implementation
    - 7.3.2.3 The `vfork()` System Call
  - 7.3.3 POSIX Threads (Pthreads)
    - 7.3.3.1 The Pthreads API
    - 7.3.3.2 Implementation of Pthread Creation
  - 7.3.4 Java Threads
  - 7.3.5 The `NSTask` Cocoa Class
  - 7.3.6 The `NSThread` Cocoa Class
  - 7.3.7 The Carbon Process Manager
  - 7.3.8 Carbon Multiprocessing Services
  - 7.3.9 The Carbon Thread Manager
- 7.4 Scheduling
  - 7.4.1 Scheduling Infrastructure Initialization
    - 7.4.1.1 Timeslicing Quantum
    - 7.4.1.2 Timing and Clocks
    - 7.4.1.3 Converting between Absolute- and Clock-Time Intervals
    - 7.4.1.4 Starting the Scheduler
    - 7.4.1.5 Retrieving the Value of the Scheduler Tick
    - 7.4.1.6 Some Periodic Kernel Activities
  - 7.4.2 Scheduler Operation
    - 7.4.2.1 Priority Ranges

- 7.4.2.2 Run Queues
- 7.4.2.3 Scheduling Information in Tasks and Threads
- 7.4.2.4 Processor Usage Accounting
- 7.4.3 Scheduling Policies
  - 7.4.3.1 `THREAD_STANDARD_POLICY`
  - 7.4.3.2 `THREAD_EXTENDED_POLICY`
  - 7.4.3.3 `THREAD_PRECEDENCE_POLICY`
  - 7.4.3.4 `THREAD_TIME_CONSTRAINT_POLICY`
  - 7.4.3.5 Priority Recomputation on Policy Change
  - 7.4.3.6 Task Roles
- 7.5 The `execve()` System Call
  - 7.5.1 Mach-O Binaries
    - 7.5.1.1 Preparations for the Execution of a Mach-O File
    - 7.5.1.2 Loading the Mach-O File
    - 7.5.1.3 Handling `Setuid` and `Setgid`
    - 7.5.1.4 Execution Notification
    - 7.5.1.5 Configuring the User Stack
    - 7.5.1.6 Finishing Up
  - 7.5.2 Fat (Universal) Binaries
  - 7.5.3 Interpreter Scripts
- 7.6 Launching Applications
  - 7.6.1 Mapping Entities to Handlers
  - 7.6.2 Uniform Type Identifiers

## **Chapter 8** Memory

- 8.1 Looking Back
  - 8.1.1 Virtual Memory and UNIX
  - 8.1.2 Virtual Memory and Personal Computing
  - 8.1.3 Roots of the Mac OS X Virtual Memory Subsystem
- 8.2 An Overview of Mac OS X Memory Management
  - 8.2.1 Reading Kernel Memory from User Space
    - 8.2.1.1 `dd` and `/dev/kmem`
    - 8.2.1.2 The `kvm(3)` Interface
  - 8.2.2 Querying Physical Memory Size
- 8.3 Mach VM
  - 8.3.1 Overview
  - 8.3.2 Task Address Spaces
  - 8.3.3 VM Maps

- 8.3.4 VM Map Entries
- 8.3.5 VM Objects
  - 8.3.5.1 Contents of a VM Object
  - 8.3.5.2 Backing Stores
- 8.3.6 Pagers
  - 8.3.6.1 External Pagers
  - 8.3.6.2 A Pager's Port
  - 8.3.6.3 The Mach Pager Interface
- 8.3.7 Copy-on-Write
- 8.3.8 The Physical Map (Pmap)
  - 8.3.8.1 The Pmap Interface
- 8.4 Resident Memory
  - 8.4.1 The `vm_page` Structure
  - 8.4.2 Searching for Resident Pages
  - 8.4.3 Resident Page Queues
  - 8.4.4 Page Replacement
  - 8.4.5 Physical Memory Bookkeeping
  - 8.4.6 Page Faults
- 8.5 Virtual Memory Initialization during Bootstrap
- 8.6 The Mach VM User-Space Interface
  - 8.6.1 `mach_vm_map()`
  - 8.6.2 `mach_vm_remap()`
  - 8.6.3 `mach_vm_allocate()`
  - 8.6.4 `mach_vm_deallocate()`
  - 8.6.5 `mach_vm_protect()`
  - 8.6.6 `mach_vm_inherit()`
  - 8.6.7 `mach_vm_read()`
  - 8.6.8 `mach_vm_write()`
  - 8.6.9 `mach_vm_copy()`
  - 8.6.10 `mach_vm_wire()`
  - 8.6.11 `mach_vm_behavior_set()`
  - 8.6.12 `mach_vm_msync()`
  - 8.6.13 Statistics
- 8.7 Using the Mach VM Interfaces
  - 8.7.1 Controlling Memory Inheritance
  - 8.7.2 Debugging the Mach VM Subsystem
  - 8.7.3 Protecting Memory
  - 8.7.4 Accessing Another Task's Memory
  - 8.7.5 Naming and Sharing Memory
- 8.8 Kernel and User Address Space Layouts

- 8.9 Universal Page Lists (UPLs)
- 8.10 Unified Buffer Cache (UBC)
  - 8.10.1 The UBC Interface
  - 8.10.2 The NFS Buffer Cache
- 8.11 The Dynamic Pager Program
- 8.12 The Update Daemon
- 8.13 System Shared Memory
  - 8.13.1 Applications of Shared Memory
  - 8.13.2 Implementation of the Shared Memory Server Subsystem
    - 8.13.2.1 `shared_region_make_private_np()`
    - 8.13.2.2 `shared_region_map_file_np()`
    - 8.13.2.3 `load_shared_file()`
    - 8.13.2.4 `reset_shared_file()`
    - 8.13.2.5 `new_system_shared_regions()`
  - 8.13.3 The Loading of Shared Object Files by the Dynamic Linker
  - 8.13.4 The Use of `shared_region_map_file_np()` by a System Application
  - 8.13.5 A Note on Prebinding
- 8.14 Task Working Set Detection and Maintenance
  - 8.14.1 The TWS Mechanism
  - 8.14.2 TWS Implementation
- 8.15 Memory Allocation in User Space
  - 8.15.1 A Historical Break
  - 8.15.2 Memory Allocator Internals
    - 8.15.2.1 Tiny Allocations
    - 8.15.2.2 Small Allocations
    - 8.15.2.3 Large Allocations
    - 8.15.2.4 Huge Allocations
  - 8.15.3 The `malloc()` Routine
  - 8.15.4 The Largest Single Allocation (32-bit)
  - 8.15.5 The Largest Single Allocation (64-bit)
  - 8.15.6 Enumerating All Pointers
  - 8.15.7 Displaying Scalable Zone Statistics
  - 8.15.8 Logging Malloc Operations
  - 8.15.9 Intercepting the Malloc Layer
- 8.16 Memory Allocation in the Kernel
  - 8.16.1 Page-Level Allocation
  - 8.16.2 `kmem_alloc`
  - 8.16.3 The Mach Zone Allocator



- 8.16.4 The Kalloc Family
- 8.16.5 The OSMalloc Family
- 8.16.6 Memory Allocation in the I/O Kit
- 8.16.7 Memory Allocation in the Kernel's BSD Portion
- 8.16.8 Memory Allocation in libkern's C++ Environment
- 8.17 Memory-Mapped Files
- 8.18 64-bit Computing
  - 8.18.1 Reasons for 64-bit Computing
    - 8.18.1.1 32-bit Execution on 64-bit PowerPC
    - 8.18.1.2 Need for Address Space
    - 8.18.1.3 Large-File Support
  - 8.18.2 Mac OS X 10.4: 64-bit User Address Spaces
    - 8.18.2.1 Data Model
    - 8.18.2.2 Implementation
    - 8.18.2.3 Usage and Caveats
  - 8.18.3 Why Not to Use 64-bit Executables
  - 8.18.4 The 64-bit "Scene"

## **Chapter 9** Interprocess Communication

- 9.1 Introduction
  - 9.1.1 The Evolution of IPC
  - 9.1.2 IPC in Mac OS X
- 9.2 Mach IPC: An Overview
  - 9.2.1 Mach Ports
    - 9.2.1.1 Ports for Communication
    - 9.2.1.2 Port Rights
    - 9.2.1.3 Ports as Objects
    - 9.2.1.4 Mach Port Allocation
  - 9.2.2 Mach IPC Messages
    - 9.2.2.1 Message Header
    - 9.2.2.2 Message Body
    - 9.2.2.3 Message Trailer
- 9.3 Mach IPC: The Mac OS X Implementation
  - 9.3.1 IPC Spaces
    - 9.3.1.1 IPC Entry Table
    - 9.3.1.2 IPC Entry Splay Tree
  - 9.3.2 The Anatomy of a Mach Port
    - 9.3.2.1 What's in a Port's Name?

- 9.3.2.2 Validity of a Port Name
- 9.3.3 Tasks and IPC
- 9.3.4 Threads and IPC
- 9.3.5 Port Allocation
- 9.3.6 Messaging Implementation
- 9.3.7 IPC Subsystem Initialization
- 9.4 Name and Bootstrap Servers
  - 9.4.1 The Network Message Server
  - 9.4.2 The Bootstrap Server
    - 9.4.2.1 The Bootstrap Port
    - 9.4.2.2 The Bootstrap Context
    - 9.4.2.3 Debugging the Bootstrap Server
  - 9.4.3 The Bootstrap Server API
    - 9.4.3.1 Displaying Information about All Known Services
    - 9.4.3.2 Creating a Crash-Resistant Server
- 9.5 Using Mach IPC
  - 9.5.1 A Simple Client-Server Example
  - 9.5.2 Dead Names
  - 9.5.3 Port Sets
  - 9.5.4 Interposition
  - 9.5.5 Transferring Out-of-Line Memory and Port Rights
- 9.6 MIG
  - 9.6.1 MIG Specification Files
  - 9.6.2 Using MIG to Create a Client-Server System
  - 9.6.3 MIG in the Kernel
    - 9.6.3.1 Interfaces to Kernel Objects
    - 9.6.3.2 MIG Initialization in the Kernel
- 9.7 Mach Exceptions
  - 9.7.1 Programmer-Visible Aspects of Mach's Exception-Handling Facility
  - 9.7.2 The Mach Exception-Handling Chain
    - 9.7.2.1 Delivering Exceptions
    - 9.7.2.2 Unresolved Kernel Traps
  - 9.7.3 Example: A Mach Exception Handler
- 9.8 Signals
  - 9.8.1 Reliability
  - 9.8.2 The Number of Signals
  - 9.8.3 Application-Defined Signals
  - 9.8.4 Signal-Based Notification of Asynchronous I/O
  - 9.8.5 Signals and Multithreading

- 9.8.6 Signal Actions
- 9.8.7 Signal Generation and Delivery
- 9.8.8 Mach Exceptions and Unix Signals Living Together
- 9.8.9 Exceptions, Signals, and Debugging
- 9.8.10 The `ptrace()` System Call
- 9.9 Pipes
- 9.10 Named Pipes (Fifos)
- 9.11 File Descriptor Passing
- 9.12 XSI IPC
- 9.13 POSIX IPC
  - 9.13.1 POSIX Semaphores
  - 9.13.2 POSIX Shared Memory
- 9.14 Distributed Objects
- 9.15 Apple Events
  - 9.15.1 Tiling Application Windows Using Apple Events in AppleScript
  - 9.15.2 Building and Sending an Apple Event in a C Program
  - 9.15.3 Causing the System to Sleep by Sending an Apple Event
- 9.16 Notifications
  - 9.16.1 Foundation Notifications
  - 9.16.2 The `notify(3)` API
  - 9.16.3 Kernel Event Notification Mechanism (`kqueue(2)`)
  - 9.16.4 Core Foundation Notifications
  - 9.16.5 Fsevents
  - 9.16.6 Kauth
- 9.17 Core Foundation IPC
  - 9.17.1 Notifications
  - 9.17.2 The Run Loop
    - 9.17.2.1 Run-Loop Observers
    - 9.17.2.2 Run-Loop Sources
      - 9.17.2.2 CFMachPort
      - 9.17.2.2 CFMessagePort
      - 9–66 CFSocket
      - 9–67 CFRunLoopTimer
- 9.18 Synchronization
  - 9.18.1 Interfaces for Atomic Operations
  - 9.18.2 Low-Level Locking
    - 9.18.2.1 Spinlocks
    - 9.18.2.2 Mutexes
    - 9.18.2.3 Read/Write Locks

- 9.18.2.4 Lock Groups and Attributes
- 9.18.3 BSD Condition Variables
- 9.18.4 Mach Lock Sets
- 9.18.5 Mach Semaphores
- 9.18.6 Pthreads Synchronization Interfaces
- 9.18.7 Locking in the I/O Kit
- 9.18.8 Funnels
  - 9.18.8.1 History
  - 9.18.8.2 Funnels in Mac OS X
- 9.18.9 SPLs
- 9.18.10 Advisory-Mode File Locking

## **Chapter 10** Extending the Kernel

- 10.1 A Driver down the Memory Lane
  - 10.1.1 Driver Programming Considered Difficult
  - 10.1.2 Good Inheritance
  - 10.1.3 Everything Is a File
  - 10.1.4 There Is More to Extending the Kernel Than Driving Devices
- 10.2 The I/O Kit
  - 10.2.1 Embedded C++
  - 10.2.2 I/O Kit Class Hierarchy
  - 10.2.3 I/O Kit Families
  - 10.2.4 I/O Kit Drivers
  - 10.2.5 Nubs
  - 10.2.6 General I/O Kit Classes
    - 10.2.6.1 Classes for Memory-Related Operations
    - 10.2.6.2 Classes for Synchronization and Serialization of Access
    - 10.2.6.3 Miscellaneous Classes
  - 10.2.7 The Work Loop
  - 10.2.8 The I/O Registry
  - 10.2.9 The I/O Catalog
  - 10.2.10 I/O Kit Initialization
  - 10.2.11 Driver Matching in the I/O Kit
- 10.3 DART
- 10.4 Dynamically Extending the Kernel
  - 10.4.1 The Structure of a Kernel Extension
  - 10.4.2 Creation of Kernel Extensions

- 10.4.3 Management of Kernel Extensions
- 10.4.4 Automatic Loading of Kernel Extensions
- 10.5 Communicating with the Kernel
- 10.6 Creating Kernel Extensions
  - 10.6.1 A Generic Kernel Extension
  - 10.6.2 Implementing Sysctl Variables Using a Generic Kext
  - 10.6.3 I/O Kit Device Driver Kext
- 10.7 A Programming Tour of the I/O Kit's Functionality
  - 10.7.1 Rotating a Framebuffer
  - 10.7.2 Accessing Framebuffer Memory
  - 10.7.3 Retrieving the List of Firmware Variables
  - 10.7.4 Retrieving Information about Loaded Kernel Extensions
  - 10.7.5 Retrieving Accelerometer Data from the Sudden Motion Sensor
  - 10.7.6 Listing PCI Devices
  - 10.7.7 Retrieving the Computer's Serial Number and Model Information
  - 10.7.8 Retrieving Temperature Sensor Readings
  - 10.7.9 Retrieving MAC Addresses of Ethernet Interfaces
  - 10.7.10 Implementing an Encrypted Disk Filter Scheme
- 10.8 Debugging
  - 10.8.1 Kernel Panics
  - 10.8.2 Remote Core Dumps
  - 10.8.3 Logging
  - 10.8.4 Debugging by Using GDB
  - 10.8.5 Debugging by Using KDB
  - 10.8.6 Miscellaneous Debugging Tools
  - 10.8.7 Stabs

## **Chapter 11** File Systems

- 11.1 Disks and Partitions
  - 11.1.1 The Apple Partitioning Scheme
  - 11.1.2 PC-Style Partitioning
  - 11.1.3 GUID-Based Partitioning
- 11.2 Disk Arbitration
  - 11.2.1 Retrieving a Disk's Description
  - 11.2.2 Participating in Disk Mounting Decisions
  - 11.2.3 Receiving Media Notifications from the I/O Kit

## xxviii Contents

- 11.3 The Implementation of Disk Devices
- 11.4 Disk Images
  - 11.4.1 Using the `hdiutil` Program
  - 11.4.2 RAM Disks
  - 11.4.3 The BSD Vnode Disk Driver
  - 11.4.4 Creating a Virtual Disk from Scratch
- 11.5 Files and File Descriptors
- 11.6 The VFS Layer
- 11.7 File System Types
  - 11.7.1 HFS Plus and HFS
  - 11.7.2 ISO 9660
  - 11.7.3 MS-DOS
  - 11.7.4 NTFS
  - 11.7.5 UDF
  - 11.7.6 UFS
  - 11.7.7 AFP
  - 11.7.8 FTP
  - 11.7.9 NFS
  - 11.7.10 SMB/CIFS
  - 11.7.11 WebDAV
  - 11.7.12 `cddafs`
  - 11.7.13 `deadfs`
  - 11.7.14 `devfs`
  - 11.7.15 `fdesc`
  - 11.7.16 `specfs` and `fifo`s
  - 11.7.17 `synthfs`
  - 11.7.18 `union`
  - 11.7.19 `volfs`
- 11.8 Spotlight
  - 11.8.1 Spotlight's Architecture
  - 11.8.2 The `Fsevents` Mechanism
  - 11.8.3 Importing Metadata
  - 11.8.4 Querying Spotlight
  - 11.8.5 Spotlight Command-Line Tools
  - 11.8.6 Overcoming Granularity Limitations
- 11.9 Access Control Lists
- 11.10 The Kauth Authorization Subsystem
  - 11.10.1 Kauth Concepts
    - 11.10.1.1 Scopes and Actions
    - 11.10.1.2 Listeners and Authorization Decisions

- 11.10.2 Implementation
- 11.10.3 A Vnode-Level File System Activity Monitor

## **Chapter 12** The HFS Plus File System

- 12.1 Analysis Tools
  - 12.1.1 HFSDDebug
  - 12.1.2 Interface for Retrieving File System Attributes
  - 12.1.3 Mac OS X Command-Line Tools
  - 12.1.4 HFS+ Source and Technical Note TN1150
- 12.2 Fundamental Concepts
  - 12.2.1 Volumes
  - 12.2.2 Allocation Blocks
  - 12.2.3 Extents
  - 12.2.4 File Forks
  - 12.2.5 Clumps
  - 12.2.6 B-Trees
    - 12.2.6.1 B+ Trees in HFS+
    - 12.2.6.2 Nodes
    - 12.2.6.3 Records
    - 12.2.6.4 Searching
- 12.3 The Structure of an HFS+ Volume
- 12.4 Reserved Areas
- 12.5 The Volume Header
  - 12.5.1 Viewing the Volume Header
  - 12.5.2 Viewing a Volume Control Block
- 12.6 The HFS Wrapper
- 12.7 Special Files
  - 12.7.1 The Allocation File
    - 12.7.1.1 Viewing the Contents of the Allocation File
    - 12.7.1.2 The Roving Next-Allocation Pointer
  - 12.7.2 The Catalog File
    - 12.7.2.1 Catalog Node IDs
    - 12.7.2.2 Examining the Catalog B-Tree
  - 12.7.3 The Extents Overflow File
    - 12.7.3.1 Examining Fragmentation
    - 12.7.3.2 Examining the Extents Overflow B-Tree
  - 12.7.4 The Attributes File
    - 12.7.4.1 Working with Extended Attributes

- 12.7.4.2 Examining the Attributes B-Tree
- 12.7.5 The Startup File
- 12.8 Examining HFS+ Features
  - 12.8.1 Case Sensitivity
  - 12.8.2 Filename Encodings
  - 12.8.3 Permissions
    - 12.8.3.1 Manipulating Volume-Level Ownership Rights
    - 12.8.3.2 Repairing Permissions
  - 12.8.4 Journaling
    - 12.8.4.1 Enabling or Disabling Journaling on a Volume
    - 12.8.4.2 Observing the Journal's Operation
  - 12.8.5 Quotas
  - 12.8.6 Hard Links
  - 12.8.7 Unlinking Open Files
  - 12.8.8 Symbolic Links
  - 12.8.9 Aliases
  - 12.8.10 Resource Forks
- 12.9 Optimizations
  - 12.9.1 On-the-Fly Defragmentation
  - 12.9.2 The Metadata Zone
  - 12.9.3 Hot File Clustering
    - 12.9.3.1 Hot File Clustering Stages
    - 12.9.3.2 The Hot Files B-Tree
    - 12.9.3.3 The Working of Hot File Clustering
- 12.10 Miscellaneous Features
  - 12.10.1 Special System Calls
  - 12.10.2 Freezing and Thawing a Volume
  - 12.10.3 Extending and Shrinking a Volume
  - 12.10.4 Volume Notifications
  - 12.10.5 Support for Sparse Devices
- 12.11 Comparing Mac OS X File Systems
- 12.12 Comparing HFS+ and NTFS

## **Appendix A** Mac OS X on x86-Based Macintosh Computers

- A.1 Hardware Differences
- A.2 Firmware and Booting
- A.3 Partitioning
- A.4 Universal Binaries



- A.5 Rosetta
- A.6 Byte Ordering
- A.7 Miscellaneous Changes
  - A.7.1 No Dual-Mapped Kernel Address Space
  - A.7.2 Nonexecutable Stack
  - A.7.3 Thread Creation
  - A.7.4 System Calls
  - A.7.5 No /dev/mem or /dev/kmem
  - A.7.6 A New I/O Kit Plane