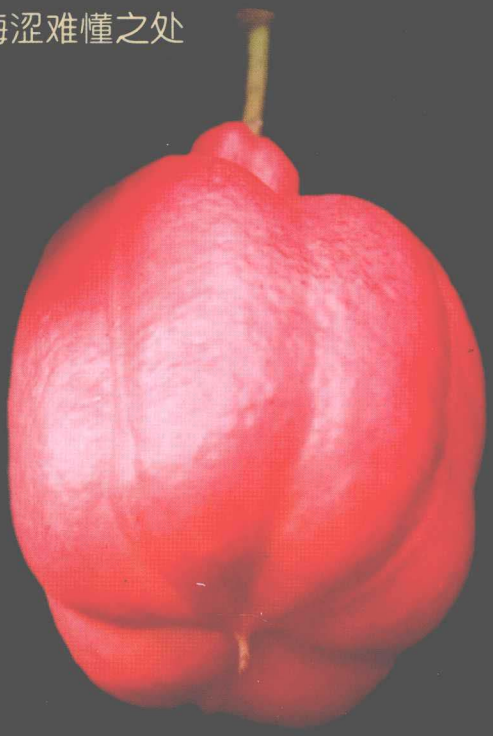
TURING 图灵程序设计丛书 移动开发系列

Apress®

- 唯一专注XNU内核开发实践的图书
- 一线设备驱动程序开发人员现身说法
- 轻松理解内核开发的晦涩难懂之处



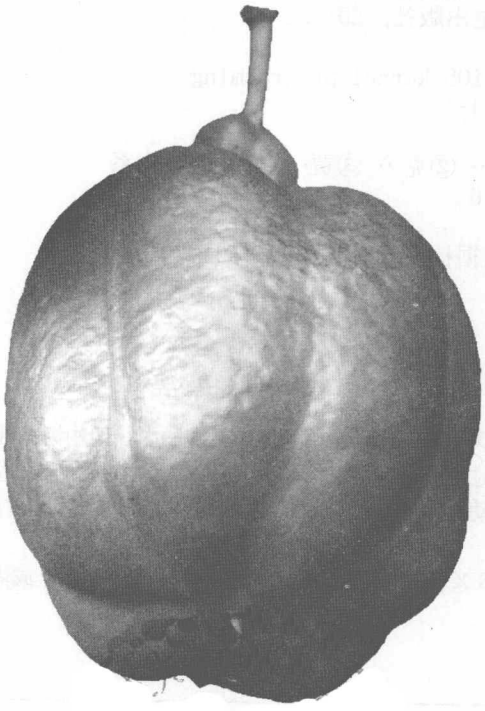
OS X and i gramming

OS X与iOS内核编程

[澳] Ole Henry Halvorsen Douglas Clarke 著
贾伟 译

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 移动开发系列



OS X and iOS Kernel Programming

OS X与iOS内核编程

[澳] Ole Henry Halvorsen Douglas Clarke 著
贾伟 译

人民邮电出版社
北京

图书在版编目（CIP）数据

OS X与iOS内核编程 / （澳）哈尔沃森（Halvorsen, O. H.），（澳）克拉克（Clarke, D.）著；贾伟译. -- 北京：人民邮电出版社，2013.6
（图灵程序设计丛书）
书名原文：OS X and iOS kernel programming
ISBN 978-7-115-31824-4

Ⅰ. ①O… Ⅱ. ①哈… ②克… ③贾… Ⅲ. ①操作系统－程序设计 Ⅳ. ①TP316

中国版本图书馆CIP数据核字(2013)第095050号

内 容 提 要

本书面向 Mac OS X 和 iOS 操作系统，介绍了操作系统和内核架构等基础知识，以及内存管理、线程同步、I/O Kit 框架等基本概念。通过最贴近实战的方法帮助读者编写高效的内核级代码。本书两位作者具备丰富的计算机硬软件以及设备驱动程序开发经验，在他们的指引下，读者定能为 USB 和 Thunderbolt 等设备开发设备驱动程序。

如果你对 iOS 和 Mac OS X 操作系统感兴趣，关注内核开发实践，或者你是驱动程序开发人员，那么本书适合你阅读。

◆ 著 [澳] Ole Henry Halvorsen Douglas Clarke
译 贾 伟
责任编辑 刘美英
执行编辑 张 敏 李 静
责任印制 焦志炜

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：25
字数：585千字 2013年6月第1版
印数：1－3 500册 2013年6月北京第1次印刷
著作权合同登记号 图字：01-2012-1813号

定价：89.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第 0021 号

版权声明

献　词

献给我的妻子 Jennifer，她也是我最好的朋友；献给我的一双儿女 Desmund 和 Isabel。

——Ole Henry Halvorsen

献给我父母，我很小就对计算机感兴趣，感谢父母一直以来对我的鼓励。

——Douglas Clarke

致　　谢

开始动笔写作本书是在我漂亮的龙凤胎 Isabel 和 Desmund 出生后不久。现在回想起来，那时真不是写作这样一本大部头图书的最佳时间。我打算另寻时间写作，但我妻子 Jennifer 一直支持我，她执意让我坚持下去。我不知道怎么感谢我的妻子，她为支持我写作这本书，承受了常人难以承受的压力。照顾一个小孩已非易事，更何况是双胞胎。有这样一位好妻子，我由衷地感到骄傲，这本书中我写作的那部分是我们两个人共同努力的结果。

刚开始时，幸好有我岳母的帮助，我妻子才能轻松一些，我也能安心完成初稿。也要感谢我岳父，他好长一段时间都没有吃到可口的家常饭。同样感谢我的妻兄，他在我们有需要的时候，给予了我们无私的帮助。还要感谢我的其他家庭成员、朋友、同事给予的帮助、鼓励以及出的点子。特别感谢我父母鼓励我走自己的路，做我感兴趣的事。

感谢 Apress 的编辑团队在写作过程中给予的指导、支持和帮助。同样感谢技术审稿人 Phil Jordan 和 Graham Lee 非常棒的指导意见，我也为他们发现细微错误的能力所折服。还要感谢 Barry Naujok、Ian Costello 和 Tim Serong 帮助我回答网络和内存管理方面的问题。同样要感谢本书另一位作者 Doug 的辛勤工作，没有他，本书不可能出版。

再次感谢帮助过我的所有人。

Ole Henry Halvorsen

前　言

内核开发是一项艰巨的任务，它与传统的用户应用程序开发完全不同。内核开发环境更多变、更复杂。由于任何问题都会对系统的稳定性、安全性及性能造成严重的后果，所以我们必须非常注意，要确保内核代码没有任何错误。本书介绍了内核编程必需的基本知识。从理论和实践两方面讲述了内核开发，并介绍了内核开发的基本概念（如虚拟内存和同步）和更多的实用知识。本书主要面向 Mac OS X 操作系统，而 iOS 操作系统也使用 XNU 内核，因此本书的理论知也适用于 iOS 操作系统。目前在内核执行环境下进行开发主要是因为需要通过实现设备驱动程序，控制内部或外部的硬件设备。因此，本书着重介绍设备驱动程序开发。在 XNU 内核中，设备驱动程序开发的主要框架是 I/O Kit，我们将详细介绍它。当然，光讲理论是很枯燥的，我们提供了一些能正常运行的代码示例，亲自动手能让你学到更多，你也能从这些示例入手，构建自己的驱动程序。希望你在阅读本书时充满乐趣，就像我们写作本书时的感觉一样。

本书读者对象

本书面向所有对苹果的 iOS 和 Mac OS X 操作系统感兴趣，并且关注内核开发实践的人，特别是驱动程序开发者。不管你是一名爱好者、学生还是专业工程师，我们都希望能为你提供感兴趣的内容。我们在关注内核程序设计和开发的同时，也将介绍许多操作系统技术理论，并详细概述 OS X 和 iOS 内核环境。本书的目的是介绍一些必要的知识，帮助你着手开发自己的内核扩展和驱动程序。为此，我们将特别介绍 I/O Kit 框架，以开发设备驱动程序及其扩展，也将介绍一些基本知识，帮你更深刻地理解 I/O Kit 和操作系统如何交互。如果你主要对 OS X 或 iOS 用户应用程序开发感兴趣，本书可能不适合你。本书不会介绍 Cocoa 应用程序或其他终端用户应用程序框架的开发。本书讲述的是内核编程主题，如基于苹果的 OS X 和 iOS 平台开发驱动程序和内核扩展程序。

读者若有一些操作系统内部构件的知识，将能更好地理解本书内容。如果你学过计算机科学或计算机工程的入门课程，那么你的起点不错。另外，为了理解本书中的例子，你至少要掌握一种编程语言。因为我们重点介绍 I/O Kit，它是采用 C++语言的子集（称为嵌入式 C++）写成的，拥有一些 C++语言（至少是 C）的经验对充分利用本书将非常有益。本书不会涉及一般的编程主题或理论，将简要介绍一些操作系统基本原理，讲述背景知识以便进一步讨论。

本书结构

下面简要说明本书各章的内容。

第 1 章，“操作系统原理”。详细介绍操作系统的功能，以及它在管理计算机硬件资源方面所起的作用。描述设备驱动程序的作用及何时需要它，并介绍内核环境编程与标准应用程序开发的不同。

第 2 章，“Mac OS X 和 iOS”。概述 XNU（Mac OS X 和 iOS 使用的内核）技术框架。

第 3 章，“Xcode 和内核开发环境”。概述苹果公司提供的面向 Mac OS X 和 iOS 的开发工具。本章结尾介绍一个简单的“Hello World”内核扩展程序。

第 4 章，“I/O Kit 框架”。介绍为 Mac OS X 提供驱动程序模型的 I/O Kit 框架，及其面向对象的体系结构。这部分内容将解释 I/O Kit 如何找到适合的设备驱动程序去管理一个硬件设备，以一个一般性的设备驱动程序为例来阐述 I/O Kit 驱动程序的基本结构。

第 5 章，“应用程序与驱动程序的交互”。解释应用程序代码如何访问内核驱动程序。举例说明如何搜索和匹配一个特定的驱动程序，以及如何安装一个通知以等待驱动程序或特殊设备的到达。还要展示应用程序如何向驱动程序发送命令并等待由驱动程序发送的事件。

第 6 章，“内存管理”。将概述内核内存管理，以及驱动程序在工作时所需的不同种类的内存。我们描述了物理及内核虚拟内存地址和用户空间内存的不同点，也将向读者介绍内存描述符和内存映射等概念。

第 7 章，“同步和线程”。将讲述同步的基本原理，以及为什么对每个内核驱动程序来说同步都是必需的。本章将讨论内核锁机制（如 `IOLock`、`IOCommandGate`）及相应的用法，还将说明在自己的线程、用户空间线程和硬件中断之间，常见的驱动程序如何使用同步机制。本章还将讨论创建内核线程和异步时钟的内核设施。

第 8 章，“USB”。将为读者介绍 USB 设备的体系结构，以及驱动程序如何与 USB 设备协同工作。本章将概述 I/O Kit USB API，以及它提供的用于枚举设备和与 USB 设备之间传送数据的类。本章也将讨论支持设备移除所需的步骤，并通过示例讲述驱动如何枚举管道等资源。

第 9 章，“PCIExpress 和 Thunderbolt”。将概述 PCI 系统架构。本章也将描述 PCI 驱动程序特有的概念，如内存映射 I/O、DMA 高速数据传输、设备中断处理等。本章还将概述 `IOPCIDevice` 类，它是 I/O Kit 为了访问和配置 PCI 设备提供的一个类。本章也将讨论较新的 Thunderbolt 的相关技术。

第 10 章，“电源管理”。将描述驱动程序需要实现的支持系统进入低电源状态（如机器休眠）的方法。本章还将讲述高级电源管理，当硬件有一段时间不使用之后，驱动程序可让其进入低电源状态。

第 11 章，“串行端口驱动程序”。将描述在 Mac OS X 上如何实现一个串行端口驱动程序。本章将介绍相关的数据结构（如循环队列）和通过阻塞 I/O 和通知事件来管理数据流的技术。本章也会展示一个用户应用程序如何枚举和访问串行端口驱动程序。

第 12 章，“音频驱动程序”。将讨论如何使用 `IOAudioFamily` 框架开发全系统的音频输入和输

出设备；举例说明一个简单的虚拟音频设备，该设备能将它的音频输出复制到音频输入。

第 13 章，“网络”。将描述如何使用 IONetworkingFamily 框架实现网络接口。本章也将讲述如何编写一个网络过滤器来过滤、阻塞和修改网络分组。本章最后将介绍一个示例，说明如何编写以太网驱动程序。

第 14 章，“存储系统”。将介绍在 Mac OS X 上支持存储设备（如硬盘和 CD）的存储器驱动程序。本章将描述各层存储栈上的驱动程序，包括如何写 RAM 磁盘、分区模式和硬盘加密的过滤器驱动程序。

第 15 章，“用户空间 USB 驱动程序”。将介绍在用户应用程序中如何完全实现某些驱动程序。本章将讲述该方法的优点，以及该方法不适用的情况。

第 16 章，“调试”。将介绍如何调试驱动程序的实践信息，以及常见的问题和易犯的错误。学完本章，读者能够根据内核崩溃报告倒推出自己的代码哪里有问题，这是内核开发人员会经常面临的任务。本章将讨论 OS X 提供的调试工具（如 GDB，即 GNU debugger）。

第 17 章，“高级内核编程”。将分析内核编程中一些更高级的主题，如使用 SSE 或浮点以及实现高级驱动程序架构。

第 18 章，“部署”。最后本书将介绍如何向最终用户发布驱动程序。本章将介绍如何使用苹果安装系统进行首次安装和升级。本章包括一些避免常见驱动程序安装问题的实践小技巧。

目　录

第 1 章　操作系统原理 ……1
1.1　操作系统的作用 ……3
1.2　进程管理 ……3
1.3　进程地址空间 ……4
1.4　操作系统服务 ……5
1.5　虚拟内存 ……6
1.6　调度 ……8
1.7　硬件和驱动程序 ……9
1.8　小结 ……11

第 2 章　Mac OS X 和 iOS ……12
2.1　XNU 内核 ……17
2.1.1　内核扩展（KEXT） ……18
2.1.2　Mach ……18
2.1.3　BSD 层 ……27
2.1.4　I/O Kit ……29
2.2　小结 ……31

第 3 章　Xcode 和内核开发环境 ……32
3.1　语言的选择：C++ ……32
3.2　Xcode ……33
3.3　“Hello World” 内核扩展 ……34
3.4　加载和卸载内核扩展 ……37
3.5　使用控制台查看输出 ……39
3.6　小结 ……40

第 4 章　I/O Kit 框架 ……41
4.1　I/O Kit 模型 ……41
4.2　对象关系 ……43
4.3　Info.plist 文件 ……43
4.3.1　驱动程序类 ……46
4.3.2　IORegistryExplorer ……50
4.4　内核库：libkern ……52
4.4.1　OSObject ……52
4.4.2　容器类 ……54
4.5　小结 ……56

第 5 章　应用程序与驱动程序的交互 ……57
5.1　I/O Kit 框架 ……58
5.2　查找驱动程序 ……58
5.3　观察设备移除 ……63
5.4　修改设备驱动程序属性 ……65
5.5　基于状态的交互 ……68
5.6　驱动程序的通知 ……79
5.7　小结 ……83

第 6 章　内存管理 ……84
6.1　内存类型 ……84
6.1.1　CPU 物理地址 ……85
6.1.2　总线物理地址 ……85
6.1.3　用户和内核虚拟地址 ……85
6.1.4　内存顺序：大端序和小端序 ……86
6.1.5　32 位和 64 位内存寻址 ……87
6.2　内存分配 ……88
6.2.1　底层分配机制 ……89
6.2.2　Mach 内存域分配器 ……89
6.2.3　kalloc 家族 ……89
6.2.4　BSD 中的内存分配 ……90
6.2.5　I/O Kit 内存分配 ……91
6.2.6　使用 C++的 new 操作符分配内存 ……92
6.3　内存描述符 ……92
6.3.1　IOBufferMemoryDescriptor ……94

6.3.2 其他的内存描述符……95
6.4 映射内存……95
6.4.1 用户空间任务到内核空间的内存映射……95
6.4.2 IOMemoryMap 类……97
6.4.3 内核到用户空间任务的内存映射……98
6.4.4 将内存映射到指定的用户空间任务……99
6.4.5 物理地址映射……100
6.5 小结……101

第 7 章 同步和线程……102
7.1 同步原语……102
7.2 原子操作……104
7.3 锁……107
7.3.1 自旋锁……107
7.3.2 互斥……109
7.3.3 条件变量……110
7.3.4 读/写互斥……112
7.4 同步异步事件：工作环……113
7.4.1 IOCommandGate……115
7.4.2 定时器……116
7.4.3 释放工作环……117
7.5 内核线程……118
7.6 小结……119

第 8 章 USB……120
8.1 USB 体系结构……120
8.1.1 USB 传输速度……122
8.1.2 主机控制器……123
8.1.3 USB 协议……124
8.1.4 端点……126
8.1.5 USB 描述符……126
8.1.6 USB 设备类……127
8.2 I/O Kit USB 支持……128
8.2.1 USB 设备和驱动程序处理……129
8.2.2 加载 USB 驱动程序……130
8.2.3 USB Prober……132
8.2.4 驱动程序示例：USB 大容量存储器设备驱动程序……133
8.2.5 驱动程序启动……138
8.2.6 处理设备移除……138
8.2.7 枚举接口……139
8.2.8 枚举端点……140
8.2.9 执行设备请求……141
8.2.10 执行批量端点和中断端点的 I/O……144
8.3 小结……147

第 9 章 PCI Express 和 Thunderbolt……148
9.1 I/O Kit 中的 PCI……150
9.1.1 匹配和加载驱动程序……151
9.1.2 驱动程序示例：一个简单的 PCI 驱动程序……154
9.1.3 访问配置空间寄存器……156
9.1.4 PCI I/O 内存区……158
9.1.5 处理设备移除……161
9.2 中断……162
9.2.1 I/O Kit 中断机制……163
9.2.2 注册接收中断……164
9.2.3 启用 MSI……166
9.2.4 处理主中断……166
9.2.5 处理二级中断……168
9.3 直接内存访问……168
9.3.1 将物理地址转换为总线地址……170
9.3.2 为 DMA 准备内存……171
9.3.3 建立分散/聚集列表……172
9.3.4 IODMACommand 类……173
9.4 小结……175

第 10 章 电源管理……177
10.1 响应电源状态改变……179
10.2 请求电源状态改变……184
10.3 处理设备空闲……185
10.4 观察设备电源状态改变……186
10.5 汇总……187
10.6 小结……191

第 11 章 串行端口驱动程序……192
11.1 Mac OS X 串行端口体系结构概览……192
11.2 串行端口驱动程序……194

11.3 实现 IOSerialDriverSync 类 ……197
11.4 串行端口状态 ……200
11.5 串行端口事件 ……204
11.6 串行数据传输 ……207
11.7 从用户空间访问串行端口 ……211
11.8 小结 ……214

第 12 章 音频驱动程序 ……215

12.1 数字音频和音频设备简介 ……215
12.2 Core Audio ……217
12.3 I/O Kit 音频支持 ……218
12.4 实现一个音频驱动程序 ……219
12.4.1 驱动程序和硬件初始化 ……221
12.4.2 注册音频控制 ……223
12.4.3 实现音频引擎 ……225
12.4.4 I/O 引擎初始化 ……226
12.4.5 其他的音频引擎功能 ……235
12.5 小结 ……236

第 13 章 网络 ……238

13.1 网络内核扩展 ……242
13.1.1 内核控制 KPI ……242
13.1.2 套接字过滤器 ……242
13.1.3 因特网协议过滤器 ……250
13.1.4 接口过滤器 ……255
13.2 调试和测试网络扩展 ……258
13.3 I/O Kit 中的网络 ……259
13.3.1 构建简单以太网控制器驱动程序 ……261
13.3.2 MyEthernetDriver 设计 ……262
13.3.3 驱动程序初始化和启动 ……264
13.3.4 介质和状态选择 ……266
13.3.5 配置设备硬件地址 ……268
13.3.6 启用和禁用设备 ……268
13.3.7 传输网络分组 ……270
13.3.8 接收分组 ……271
13.3.9 对 MyEthernetDriver 进行测试 ……274
13.4 小结 ……276

第 14 章 存储系统 ……277

14.1 传输层驱动程序 ……278
14.2 IOBlockStorageDevice 接口 ……279
14.3 构建 RAM 磁盘设备 ……282
14.4 分区规则 ……292
14.4.1 实现一个示例分区规则 ……293
14.4.2 媒介内容线索属性 ……300
14.5 媒介过滤器驱动程序 ……300
14.5.1 加密过滤器规则示例 ……302
14.5.2 创建一个自定义 GUID 分区表 ……306
14.6 小结 ……308

第 15 章 用户空间 USB 驱动程序 ……310

15.1 背景 ……310
15.2 IOUSBLib 框架 ……311
15.3 处理异步操作 ……315
15.4 IOUSBDeviceInterface 类 ……316
15.5 IOUSBInterfaceInterface 类 ……320
15.5.1 属性方法 ……321
15.5.2 端点数据传输方法 ……322
15.5.3 低延迟同步传输 ……328
15.6 小结 ……330

第 16 章 调试 ……331

16.1 常见的问题类型 ……331
16.2 内核恐慌 ……332
16.3 调试机制 ……333
16.3.1 启动时修复崩溃 ……335
16.3.2 采用 IOLog()追踪 ……335
16.3.3 输出栈跟踪 ……336
16.3.4 通过 FireWire 远程跟踪 ……337
16.3.5 远程内核核心转储 ……339
16.3.6 KDB ……340
16.3.7 通过以太网或 FireWire 使用 GDB 远程调试 ……340
16.3.8 实时调试运行的内核 ……344
16.3.9 使用虚拟机调试 ……344
16.3.10 在内核中使用 GDB 调试 ……344

16.3.11 使用 Activity Monitor 诊断挂起进程 ····· 354
16.3.12 查找内存和资源泄漏 ····· 355
16.4 小结 ····· 356
第 17 章 高级内核编程 ····· 357
17.1 内核中的 SSE 和浮点 ····· 357
17.2 多功能驱动程序 ····· 358
17.3 编写 I/O Kit 族 ····· 358
17.4 内核控制 KPI ····· 359
17.4.1 内核控制注册 ····· 361
17.4.2 客户端连接 ····· 362
17.4.3 获取和设置选项 ····· 363
17.4.4 从用户空间访问内核控制 ····· 364
17.5 内核中的进程处理 ····· 365
17.6 加载资源 ····· 366
17.7 KEXT 资源之外的内容 ····· 367
17.8 内核驱动程序通知 ····· 368
17.9 小结 ····· 371
第 18 章 部署 ····· 372
18.1 安装和加载内核扩展 ····· 372
18.2 加载首选项和设置 ····· 374
18.3 内核扩展的版本管理 ····· 375
18.4 测试和品质保证 ····· 375
18.5 打包 KEXT 和软件 ····· 376
18.5.1 构建 Hello World 内核扩展安装包 ····· 378
18.5.2 将内容添加至安装包 ····· 378
18.5.3 配置安装包 ····· 379
18.5.4 构建安装包 ····· 382
18.5.5 卸载安装包 ····· 383
18.6 小结 ····· 383
索引 ····· 385

第1章 操作系统原理

操作系统的作用是为用户提供应用软件的运行环境。用户运行应用程序时所执行的任务依赖于操作系统提供的服务，然而，操作系统执行任务时，许多情况下都无需用户甚至程序员的干预。例如，应用程序从硬盘读取文件时，程序员只需调用操作系统提供的函数即可，执行读取的特定步骤完全交由操作系统处理。程序员不用了解从计算机硬盘中读取文件与从外部USB闪存中读取文件有什么不同，这是操作系统关心的事情。

绝大多数程序员都开发过很多用户使用的代码，他们可能还会使用框架（如Cocoa）提供图形化用户界面与用户进行交互。Mac或iPhone应用商店提供的所有应用程序都属于这一类。本书讲述的不是应用软件开发，而是内核扩展开发，即编写为应用程序提供服务的代码。下列两种情况非常适合进行内核扩展：操作系统与自定义的硬件设备协同工作时，以及添加新文件系统的支持时。例如，经过内核扩展后，iTunes就可以使用新的USB音频设备，以太网卡就可以为网络应用程序提供接口，如图1-1所示。对文件系统进行内核扩展，就可以将在Windows计算机上格式化后的硬盘装载在Mac机上，使其看起来像一个标准的Mac驱动。

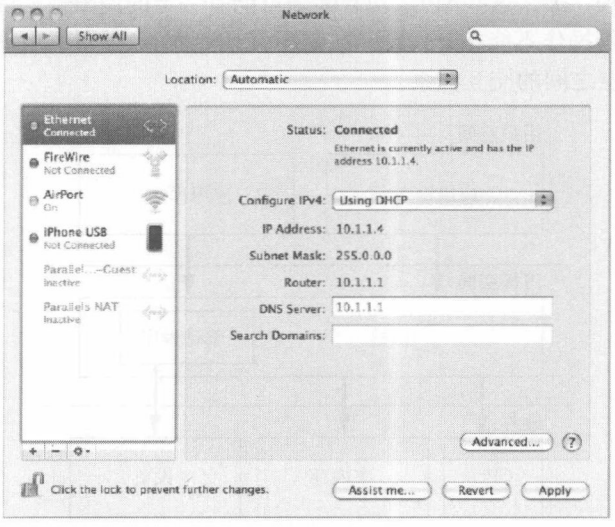


图1-1 Mac OS X系统网络界面的首选项表示网络内核扩展

操作系统的一项重要功能就是管理计算机硬件资源（如内存与CPU）以及外部设备（如磁盘存储器与键盘）。操作系统所支持的硬件设备的集合在不同的机器上有很大差别。尽管MacBook Air与Mac Pro运行相同的操作系统，但它们的硬件配置却有很大不同。为了让操作系统支持多种硬件配置而又不变得膨胀，支持每个硬件组件的代码被分别打包成一种特殊类型的内核扩展程序（称为驱动程序）。这种模块化方式使操作系统可以根据系统上现有的硬件按需加载驱动程序。厂商因而也可以在系统中安装相应的驱动程序以支持他们自己的硬件。安装标准的Mac OS X系统需要装100多个驱动程序，但其中只有一个子集用于运行特定的系统。

内核扩展程序开发与应用程序开发有很大不同。应用程序一般由用户事件驱动执行。应用程序在用户启动它时开始运行，然后等待用户点击按钮或选择菜单项，再处理相应的请求。相反，内核扩展没有用户界面，无需与用户交互。它们由操作系统加载，并由操作系统调用去执行自身无法独立完成的任务（如访问由内核扩展驱动的硬件设备）。

为了提高系统的安全性和稳定性，现代操作系统（如Mac OS X）将操作系统核心代码（内核）与应用程序、用户运行的服务程序分离。任何作为内核的一部分运行的代码（如驱动代码），都要在“内核空间”中运行。运行在内核空间中的代码被授予特权，如可以直接读写连接到计算机上的硬件设备，但标准的用户应用程序不享有该特权。

相反，用户处理的标准应用程序代码要在“用户空间”中运行。用户空间中运行的软件无法直接访问硬件。因此，为了访问硬件，用户代码必须向内核发送请求（如读取磁盘的请求），让内核代表应用程序执行任务。

用户空间与内核空间中运行的代码之间具有明确的界限。应用程序只能通过调用操作系统发布给用户空间代码的函数访问内核。类似地，内核空间代码独立运行于用户空间代码环境之外。内核不使用与用户空间代码相同的富编程API，它有自己的API集，供内核扩展开发者使用。如果你熟悉用户空间编程，那么这些API最初看起来可能有一定的局限性。因为通常情况下，用户交互和访问文件系统等操作不能在开发内核扩展时使用。图1-2说明了用户空间代码和内核空间代码的划分，以及各层之间的交互关系。

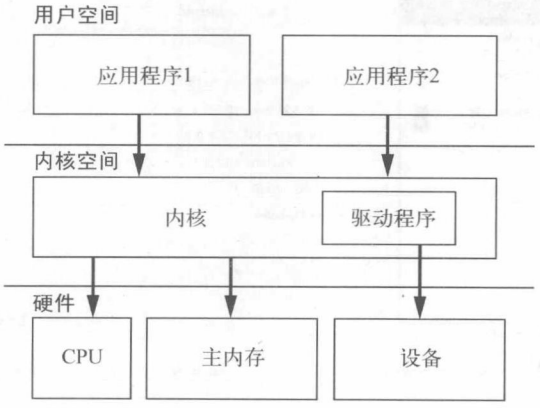


图1-2　现代操作系统职责分层

强制应用程序向内核发送访问硬件的请求有一大优点，即内核（及内核驱动程序）成为硬件设备的中央仲裁器。以声卡为例，系统上可能有多个应用程序在同一时刻播放音频，但由于所有的请求只通过一个音频驱动传送，所以该驱动程序能将来自所有应用程序的音频流混合，然后将产生的混合流提供给声卡。

本章的剩余部分将概述操作系统内核的功能，着重讲述其为用户应用程序提供硬件访问的重要性。我们将从最上层的应用软件着手，然后深入到操作系统内核层，最后介绍底层的硬件驱动程序。如果你已经熟悉这些概念，可以直接跳到第2章继续阅读。

1.1 操作系统的作用

作为启动序列的一部分，操作系统决定系统的硬件配置，搜索连接到USB接口或插入PCI扩展槽的外部设备，对它们进行初始化，如有必要，在初始化时加载驱动程序。

操作系统完成加载后，用户便能运行应用软件。应用软件可能需要分配内存或向磁盘写入文件，这些请求均由操作系统处理。对用户而言，操作系统的参与基本上是透明的。

操作系统在运行的应用程序和物理硬件之间提供了一个抽象层。应用程序一般通过向操作系统发出高层请求与硬件通信。因为这些请求由操作系统处理，所以应用程序可能完全不知道其运行环境的硬件配置（如RAM的数量、磁盘存储器是内部SSD还是外部USB驱动器）。

这个抽象层可以使应用软件在多种不同的硬件配置环境下运行，无需程序员为每种硬件添加支持，即便那些在程序发布后才出现的新硬件设备也不需要。

应用程序开发者通常可以忽略计算机系统工作的诸多细节，因为操作系统将运行应用程序的硬件平台抽象了出来。但是，作为一名驱动程序开发者，你所编写的代码将作为操作系统的一部分，直接与计算机硬件交互；你不能对操作系统的内部工作原理一无所知。为此，了解操作系统如何执行任务非常必要。

1.2 进程管理

通常，用户在计算机上安装了许多应用程序，而且这些应用程序都是纯被动实体。磁盘上的程序所包含的数据仅在程序运行时需要，这些数据由可执行代码和应用程序数据组成。当用户启动应用程序时，操作系统会将程序的代码和数据从磁盘加载到内存中，并开始执行代码。一个正在执行的程序称为“进程”。与程序不同，进程是主动实体，由程序执行期间单个实例的状态快照组成。其中包括程序代码、程序分配的内存以及当前的执行状态，如程序当前正在执行的函数的CPU指令、变量内容和内存分配情况。

通常，系统会同时运行许多进程。其中包括用户启动的应用程序（如iTunes或Safari）、操作系统自动启动的进程和不提示用户而运行的进程。例如，时间机器（Time Machine）备份服务每小时在后台自动运行一次，执行数据备份。而同一时刻，可能有同一程序的多个实例在执行，此时操作系统会将每个实例看做不同的进程。图1-3展示了Mac OS X中的实用程序Activity Monitor，我们可以用它查看系统上运行的所有进程。

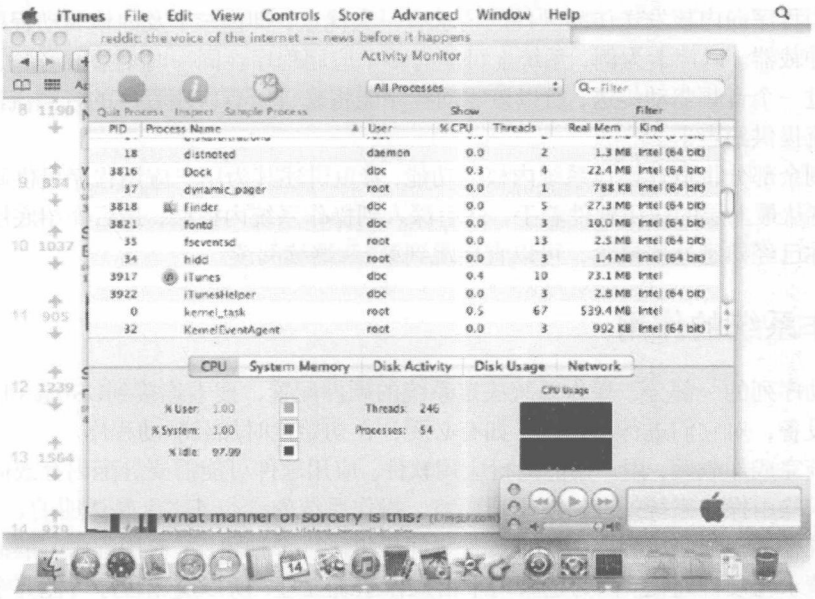


图1-3　Mac OS X上的Activity Monitor显示了系统上运行的所有进程，可以跟显示可见用户应用程序的Dock对照着看

1.3　进程地址空间

尽管任意时刻会有许多进程同时运行，但每个进程都觉察不到系统有其他进程在运行。实际上，在没有显式代码的情况下，一个进程无法与另一个进程交互，也不能对其行为产生影响。

操作系统为每个进程提供了一段可操作的内存，我们称之为进程的地址空间。地址空间是动态的，它在进程执行期间随其内存分配而发生变化。如果一个进程试图读/写其地址空间外的内存地址，操作系统通常会将其终止，并通知用户该应用程序已经崩溃。

尽管受保护的内存并非新生事物，但也是近十年才出现在用户的桌面系统中。在Mac OS X之前，Mac OS 9上运行的进程可以读/写任何内存地址，即使该地址是由另一个进程分配的缓冲区或属于操作系统本身。

没有内存保护，应用程序可以绕过操作系统，直接修改不同进程的内存和变量（无论是否得到该进程的允许），以实现进程间的通信。对操作系统结构来说也是如此。例如，Mac OS 9有一个内部全局变量，它包含每个打开的GUI窗口链表。尽管该链表名义上由操作系统拥有和操作，但实际上应用程序无须对操作系统进行任何调用，就可以遍历和修改该链表。

没有内存保护，操作系统容易受用户应用程序中bug的影响。应用程序在有内存保护的系统上运行时，最坏的情况是损坏自身的内存和结构，但这种损坏仅局限于应用程序本身。在没有内存保护的系统上（如Mac OS 9），应用程序中的程序错误还有可能覆盖操作系统的内部结构，使系统完全崩溃，需要重启才能恢复。