Robert Love

Third Edition

Linux Kernel Development

A thorough guide to the design and implementation of the Linux kernel

Developer's Library

Linux Kernel Development

Third Edition

Robert Love

Translated by: Kevin Zhang

2010-12-10 – 2011-03-15(TODO)

目录

序.....4
前言.....6
如何使用本书.....6
内核版本.....7
本书读者.....7
第三版鸣谢.....8
关于作者.....8
第 1 章 Linux 内核介绍.....10
Unix 的历史.....10
第 2 章 内核入门.....18

序

随着 Linux 内核和使用内核的应用得到越来越广泛的使用，希望参与开发和维护 Linux 的系统软件开发者的数量也在不断增加。这些工程师中有的单纯是个人兴趣，有些任职于 Linux 公司，有些为硬件厂商工作，也有一些是个人开发项目相关。

但是所有人都面临同样的一个问题：内核学习曲线越来越长、越来越陡峭。内核系统非常庞大，而且复杂度与日俱增。随着这几年的发展，当前内核开发团队的成员对内核内部的知识更加深入和广泛，这也进一步扩大了他们和新手之间的差距。

我相信 Linux 内核源码的这种可达性衰退已经成为影响内核质量的一个问题，并且随着时间的发展会越来越严重。所有关心 Linux 发展的人都希望为内核贡献的开发者数量能够得到增长。

解决这个问题的一个办法是保持代码的清晰：合理的接口、统一的层次架构、“只做一件事并且做好”等等。这也正是 Linus Torvalds 所采取的办法。

我个人建议的办法是增加代码注释：读者在阅读代码时可以据此来理解代码编写者的真实意图。（找出意图和实现之间分歧的过程被称为调试，如果意图不明确就非常难以调试）。

但是代码注释也不能提供主要子系统的更高层次的意图，以及开发者是如何理解它的。要理解这种意图，就只有相关的书籍才能做到了。

Robert Love 的这本书给我们展示了富有经验的开发者对于内核的核心认识，内核子系统提供的各种服务、它们是如何提供这些服务。所有这些知识能够满足许多不同的人：好奇者、应用开发者、想要体会内核设计的人等等。

本书还能帮助有抱负的内核开发者步入下一个层次：修改内核来完成明确的既定目标。我非常鼓励有抱负的开发者亲自动手。理解内核的最佳途径就是修改它。修改内核要求开发者对内核的理解，要站在比阅读代码更高的层次。认真的内核开发者会加入邮件列表、并与其他开发者积极沟通。这种沟通是内核贡献者互相学习和保持一致的主要动力。Robert 很好的阐述了内核的结构和相关文化。

从 Robert 的书中你能学习并得到享受，你会决定进入下一个层次，成为内核开发社区的一份子。我们通过人们的贡献来评价和衡量一个人，当你为 Linux 做出贡献时，即使你完成的工作很小，但是能够立即让数百万人获益。这是特权和责任能够得到的最大享受。

Andrew Morton

前言

当我最早考虑将我对 Linux 内核的经验转换为一本书时，我始终都怀着一丝忧虑。怎样才能使我的书成为该主题最好的书籍？如果我不能做特别的、最好的工作，那我就不会感兴趣。

最终我意识到我可以对内核这一主题提供独到的方法。我的工作是 hacking 内核、我的嗜好是 hacking 内核、我也热爱 hacking 内核。这么多年以来，我积累了大量有趣的轶闻和内情。以我的经验可以写一本如何 hack 内核（同样重要的是如何防止 hack 内核）的书。首先也是最重要的，这是一本关于 Linux 内核设计和实现的书。这本书使用的方式与它的竞争者不同，我会提供足够的信息使你能更好地完成工作。我是一个实用主义的工程师，这也是一本注重实践的书。本书应该很有趣、阅读很轻松、并且非常有用。

我希望读者通过阅读本书，能够加深对 Linux 内核规则的理解。我的目标是让你（没有阅读过本书和内核源代码的人），能迅速入门并开始编写有用的、正确的、清晰的内核代码。当然，你也可以仅仅把本书当作娱乐使用。

本书第一版出版至今，已经好几年过去了。第三版比之前两版提供了相当多的内容：修订、更新、和许多全新的章节。第三版吸收并增加了第二版之后内核的所有变更。更重要的是，Linux 内核社区决定短期内不继续进行 2.7 版本内核的开发，而是计划继续开发和稳定 2.6 系列。这个决定有许多影响，但是与本书相关的则是继续使用 2.6 版本的内核。由于 Linux 内核非常成熟，即使是内核快照也能保持相当长时间的稳定。本书可以说是内核的权威文档，同时兼顾内核的历史和未来。

如何使用本书

开发内核代码不需要天才和魔法，甚至 Unix 黑客标志性的大胡子也不需要。内核尽管也有一些自己的有趣的规则，但它与其它任何一个大型软件并无太大区别。当然你需要掌握许多细节（和任何大型项目一样），但区别只是数量，而不是性质。

要学习内核，使用源码是必须的。Linux 系统源码的开放可用性，是一份珍贵的礼物。但是仅仅阅读源码是远远不够的，你还需要深入进去并修改一些代码、找出 bug 并修复它、改进你使用的硬件驱动、增加新功能。即使是很小的事情，你都要找出并完成它。只有当你真正编写了内核代码，才能真正理解内核的所有。

内核版本

本书基于 Linux 内核 2.6 系列版本，不涉及更老的内核，除非与历史相关。例如只要对于我们的讨论是有帮助的，我们也会讨论某个内核子系统在 2.4 版本中的实现方式。具体来说，本书第三版使用了 Linux 内核 2.6.34。尽管内核一直在不断的前进，要保持更新确实很难，但我的目标是使本书能够同时适用于老的和新的内核开发者和用户。

尽管本书讨论 2.6.34 内核，但我花了很大力气确保书中的材料也能适用于 2.6.32 内核。后者是许多 Linux 发布版为“企业”内核所做的剪裁，在许多年内都将在产品系统中使用，也会有持续动态的开发支持（2.6.9，2.6.18 和 2.6.27 都是类似的长期发布版）。

本书读者

本书的目标读者是对理解 Linux 内核感兴趣的开发者和用户。它不是对内核源码一行一行的注释，也不是驱动开发的指南，或内核 API 的参考手册。相反本书的目标是提供 Linux 内核设计和实现的足够的信息，使程序员能开始开发内核代码。内核开发非常有趣，报酬也不错，而我希望本书能尽快把读者引入这个世界。本书既讨论理论又兼顾实用，所以应该同时适用于学院派和实践派的读者。我一直认为只有掌握了理论，才能更好地进行实践，所以我试图在本书中平衡二者。我希望无论你理解 Linux 内核的动机是什么，本书都能满足你的需求，为你解释清楚内核的设计和实现。

因此本书同时讲解了核心内核子系统的使用，和它们的设计与实现。我认为这一点很重要，值得我们稍加讨论。以第 8 章“Bottom Halves 和 Deferring Work”

为例，该章讲解了一个被称为 bottom halves 的设备驱动组件。在那一章中，我既讨论了内核 bottom-half 机制的设计和实现（核心内核开发者和学院派应该会感兴趣），也讨论了如何使用导出的接口来实现你自己的 bottom half（设备驱动开发者和黑客会感兴趣）。核心内核开发者当然需要理解内核的内部工作原理，而他们一般都对接口的使用非常熟悉。同时设备驱动开发者也能从理解接口背后的实现得到很多好处。

这和学习某个库的 API 与学习库的实现是一个道理。表面看来，应用程序员只需要理解 API（将接口视作黑箱通常是足够的）；库开发者只需要关心库的设计和实现。但我认为双方都应该花费一些时间来学习另一方面。那些更好地理解了底层操作系统的应用程序员，能够更好地使用操作系统的接口。同样，库开发者也不应该与应用如何使用库完全脱离。因此，我同时讨论了内核子系统的设计和使用，希望本书能够对双方都有用。

我假设读者了解 C 编程语言，熟悉 Linux 系统。有一些操作系统设计和计算机科学相关的理论对学习本书会有帮助，我也会尽可能多地解释相关的概念。如果我没有，附录的参考书目列出了一些操作系统设计方面的卓越书籍。

本书也适用于操作系统设计课程的大学教材，可以作为理论介绍教材的补充材料。本书可以在大学高年级课程或研究生课程中使用。

第三版鸣谢

（略）

关于作者

Robert Love 是一个开源程序员，演讲者，和作者，他使用和贡献 Linux 已经超过 15 年。Robert 目前是 Google 的高级软件工程师，是 Android 移动平台内核团队的成员之一。在去 Google 之前，他是 Novell Linux 桌面的首席架构师。在 Novell 之前，他是 MontaVista 软件和 Ximian 的内核工程师。

Robert 的内核项目包括抢先式内核、进程调度器、内核事件层、inotify、VM

增加、和几个设备驱动。

Robert 进行了无数演讲，撰写了许多关于 Linux 内核的文章。他也是 Linux Journal 的编辑之一。他编写的其它几本书包括 Linux System Programming 和 Linux in a Nutshell。

Robert 获得了福罗里达大学的数学和计算机科学学位。现居住于波士顿。

第 1 章 Linux 内核介绍

本章介绍 Linux 内核和 Linux 操作系统，将它们跟 Unix 的历史结合在一起。今天，Unix 指的是一组操作系统，它们实现了类似的应用编程接口（API），并且遵循相同的设计原则。但是 Unix 也是一个特定的操作系统，距今已有 40 多年。要理解 Linux，我们必须首先讨论最初的 Unix 系统。

Unix 的历史

经过四十多年的使用，计算机科学家仍然赞誉 Unix 操作系统是当今最强大最优美的系统。从 1969 年 Unix 诞生以来，Dennis Ritchie 和 Ken Thompson 的作品已经成为一个传奇，Unix 的设计经受住了时间的考验。

Unix 起源于 Multics，后者是 Bell 实验室参与的一个失败的多用户操作系统项目。Multics 项目终止以后，Bell 实验室计算机科学研究中心的成员发现自己没有可用的交互式操作系统。于是在 1969 年夏，Bell 实验室的程序员们设计出了一个文件系统，最终演化成了 Unix。为了测试该系统的设计，Thompson 在一台 PDP-7 机器上实现了这个新系统。在 1971 年，Unix 成功移植到 PDP-11；然后在 1973 年，又用 C 语言重写了 Unix 操作系统。在当时这是史无前例的一步，但也正是如此，才为将来的可移植性铺平了道路。第一个在 Bell 实验室之外得到广泛使用的 Unix 操作系统是 Unix System 第六版，经常被称为 V6。

许多公司将 Unix 移植到新机器上。这些移植在增强 Unix 功能的同时，也导致了 Unix 操作系统的几个不同变种。1977 年 Bell 实验室组合这些变种到一起，发布了 Unix System III；1982 年 AT&T 发布了 System V。（System IV 是一个内部开发版本）。

Unix 设计的简洁性，加上它又以源码的方式发布，吸引了许多外部组织的进一步开发。其中最有影响的莫过于加利福尼亚大学伯克利分校。伯克利的 Unix 变种被称为 Berkeley Software Distributions，或者 BSD。伯克利在 1977 年第一次发布 1BSD，在 Bell 实验室的 Unix 上增加了补丁和额外的软件包。随后 1978 年的 2BSD 仍然是以这种形式发布，增加了 csh 和 vi 等实用工具，这些工具一直持

续使用到今天。第一个独立的 Berkeley Unix 是 1979 年发布的 3BSD，它在已经非常丰富的特性集中增加了虚拟机功能。随后发布的是 4 系列的 BSD，4.0BSD、4.1BSD、4.2BSD、4.3BSD。这些版本增加了任务控制、动态页面、和 TCP/IP。1994 年伯克利发布了最后一个官方版本的 Berkeley Unix，重写了 VM 子系统，是为 4.4BSD。今天我们要感谢 BSD 宽容的许可，正因为此才有了后来的 Darwin、FreeBSD、NetBSD、和 OpenBSD 系统。

1980 至 1990 年间，多个工作站和服务器公司发布了自己商业版本的 Unix。这些系统都是基于 AT&T 或 Berkeley 的 Unix 发布版，但是支持自己特定硬件体系架构的高端特性。比较有名的有 Digital 的 Tru64、Hewlett Packard 的 HP-UX、IBM 的 AIX、Sequent 的 DYNIX/ptx、SGI 的 IRIX、以及 Sun 的 Solaris & SunOS。

Unix 系统最初优雅的设计，加上多年来的改革和发展，已经成为了功能强大、健壮、和稳定的操作系统。Unix 拥有许多优秀的特性。首先 Unix 非常简单，比起某些操作系统动辄几千个系统调用和不清晰的设计目标，Unix 只实现了大约几百个系统调用，而且拥有非常直接和基本的设计。第二、在 Unix 中，“一切都是文件”（好吧，也不是所有，但大多数都被表示为文件。Sockets 是明显的一个例外。最新的一些系统例如 Bell 实验室的 Unix 继承者 Plan9，几乎系统的所有方面都实现为文件）。这样就可以把所有操作数据和设备简化为一组核心系统调用：open(), read(), write(), lseek(), close()。第三、Unix 内核和相关的系统工具用 C 语言编写，这赋予了 Unix 极大的可移植性，使其适用于各种硬件体系架构，并且能够对许多开发者可用。第四、Unix 拥有非常快速的进程创建和独特的 fork()系统调用。最后、Unix 提供一组简单但却健壮的进程间通讯（IPC）原语，与快速进程创建结合在一起，允许创建简单的程序“做一件事并做到最好”。这些单一目的的程序可以被串连在一起，来完成复杂的任务。Unix 系统也因此获得了清晰的层次架构，在策略和机制之间建立了给力的分离。

今天 Unix 是一个支持抢先式多任务、多线程、虚拟机、动态页面、动态装载共享库、和 TCP/IP 网络的现代操作系统。各种 Unix 变种可以运行在大到数百个处理器的机器、小至嵌入式设备中。尽管 Unix 已经不再是一个研究项目，Unix 系统仍然在推动操作系统设计的发展，同时又保持实践性和通用操作系统的特点。

Unix 的成功归因于简单性和优雅的设计。它今天的强大起源于 Dennis Ritchie、Ken Thompson 和其它早期开发者所做出的英明决策：赋予了 Unix 不断进化却又不需妥协自己的能力。

Linus 和 Linux 介绍

Linus Torvalds 在 1991 年为一台 Intel 80386 微处理器的计算机开发了 Linux 操作系统的第一个版本，当时 80386 是一款很新而且很高级的处理器。Linus 当时是赫尔辛基大学的一名学生，正苦恼于缺乏强大而免费的 Unix 系统。当时主流的个人计算机 OS 是 Microsoft DOS，对于 Torvalds 来说只能玩波斯王子而已。Linus 也用过 Minix，一个廉价的为教学目的而产生的 Unix，但是由于 Minix 的许可和其作者所做的设计决策，导致 Linus 无法修改和发布 Minix 系统的源码，这让他感到很气馁。

面对如此困局，Linus 做了任何一个正常的大学生都会做的事情：他决定编写自己的操作系统。Linus 从编写一个简单的终端模拟器开始，他用来连接学校的大型 Unix 系统。随着大学课程的进行，他的终端模拟器得到了不断的发展和改进。不久以后，Linus 手上就有了一个粗糙但羽翼丰满的 Unix。他在 1991 年底在因特网上发布了第一个早期版本。

早期 Linux 发布版很快地获得了许多用户，但是比起最初的成功，更重要的是 Linux 很快地吸引了大批开发者——增加、修改、改进代码的黑客。而 Linux 许可的条款也使其很快就演变成为一个许多人合作开发的项目。

今天 Linux 已经是一个羽翼丰满的操作系统，可以支持 Alpha、ARM、PowerPC、SPARC、x86-64 以及许多其它体系架构。Linux 运行在小到手表，大到整间房子的超级计算机集群的不同系统中。Linux 为最小的消费电子产品和大型数据中心提供动力。今天 Linux 的商业利益也非常强劲。新的 Linux 公司如 RedHat、以及现有的计算机大企业如 IBM，都为嵌入式、移动、桌面、和服务器等领域提供基于 Linux 的解决方案。

Linux 是一个类 Unix 的系统，但它不是 Unix。也就是说尽管 Linux 借鉴了 Unix 很多的思想，也实现了 Unix API（由 POSIX 和 Single Unix Specification 定义），但

它不像其它 Unix 系统一样直接继承自 Unix 的源代码。当需要时 Linux 也会偏离 Unix 的路线，但它并没有抛弃 Unix 的设计目标，也没有破坏标准化的应用接口。

Linux 最有趣的特点之一是：它不是一个商业产品，相反，它是一个基于互联网的合作开发项目。虽然 Linus 仍然是 Linux 内核的创建者和维护者，内核的持续发展却是通过一个松散组织的开发者团队来实现。任何人都可以为 Linux 贡献。Linux 内核与系统的大部分都是免费或开放源码软件。具体来说，Linux 内核采用 GNU 通用公共许可证（GPL）版本 2.0。因此你可以自由地下载源代码，并进行任何你想要的修改。唯一需要注意的是，如果要发布你的更改，你必须提供你享受到的所有权利，包括源代码的可用性。

Linux 对很多人来说意味着很多东西。Linux 系统的基础是内核、C 库、工具链、基本的系统工具（如登录过程和 Shell 等）。Linux 系统也可以包含一个现代的 X Window 系统，包括一个全功能的桌面环境，比如 GNOME。Linux 系统拥有成千上万的免费和商业应用。在这本书中，我说的 Linux 通常指的是 Linux 内核。如果存在混淆，我会明确地指出是指一个完整的 Linux 系统，还是指 Linux 内核。严格来说，Linux 这个词仅仅表示 Linux 内核。

操作系统和内核概述

由于持续不断增长的特性，以及某些现代商业操作系统病态的设计，操作系统的概念已经很难精确定义。许多用户认为他们在屏幕上看到就是操作系统。技术上来讲，操作系统是负责实现基本使用和管理的那部分系统，本书也采纳这个观点。这包括内核和设备驱动、启动引导器、命令行 shell 或其它用户界面、以及基本的文件和系统工具。操作系统只是你需要的那些东西——不是网页浏览器或音乐播放器。系统这个术语则表示操作系统和所有运行在其上的应用程序。

当然本书的主题是内核，而用户界面是操作系统的最外层部分，内核则处于最内层。它是系统的内部核心，为系统的所有其它部分提供核心服务，管理硬件，并分配系统资源。内核有时候也被称为操作系统的 supervisor、core 或 internals。内核的典型组成包括服务中断请求的中断处理器、多个进程共享处理器时间的调度器、管理进程地址空间的内存管理子系统、以及网络和进程间通讯等系统服务。

在拥有受保护内存管理单元的现代系统中，内核一般都处于高级系统状态，区别于普通的用户应用。这包括一个受保护的内存空间，和硬件的完全访问权。这个系统状态和内存空间合起来称为内核空间。反之，用户应用则在用户空间中执行。它们只能看到机器可用资源的一个子集，只能执行特定的系统函数，只能访问内核分配给它们的硬件和内存，否则将被拒绝。当执行内核代码时，系统将以内核模式运行在内核空间中；当运行一个普通的进程时，系统就以用户模式运行在用户空间中。

应用程序通过系统调用（如图 1.1）与内核进行交互。应用通常调用系统库提供的函数（例如 C 库），而库则依赖于系统调用接口指示内核来完成应用请求的任务。有一些库还提供很多系统调用中没有的特性，因此在很多功能的实现中调用内核只是其中的一部分。例如我们熟悉的 printf()函数，它提供数据的缓存和格式化输出，它进行了很多的工作，但只是最后一步才调用了 write()把数据写入到控制台。反过来，有一些库调用则与内核系统调用有一对一的关系。例如 open()库函数基本上只是调用 open()系统调用。还有其它一些 C 库函数，如 strcpy()，则完全不使用内核。当应用执行系统调用时，我们就说内核正在代表应用程序执行（kernel is executing on behalf of the application）。此时应用就称为正在内核空间中执行系统调用（executing a system call in kernel-space），而内核则运行在进程上下文中。这种关系（应用通过系统调用接口进入内核）是应用完成工作的基本模式。

内核也管理系统的硬件。几乎 Linux 支持的所有体系架构和系统都提供中断的概念。当硬件需要与系统交互时，它就会向处理器发出一个中断，进而中断内核。中断用一个数字来标识，内核通过这个数字来执行特定的中断处理器，来处理和响应中断。例如当你打字时，键盘控制器就会产生一个中断，通知系统键盘缓冲区有了新的数据。内核拿到这个中断数值后，就可以执行正确的中断处理器。中断处理器处理键盘数据，并且通知键盘可以继续接收数据。为了同步，内核可以禁止中断（所有中断或某个特定的中断）。在许多操作系统中，包括 Linux，中断处理器并不运行在进程上下文中。相反，它们运行在特殊的中断上下文，与任何进程都无关。这个特殊的上下文存在的目的是让中断处理器能够迅速地响应中

断，然后退出。

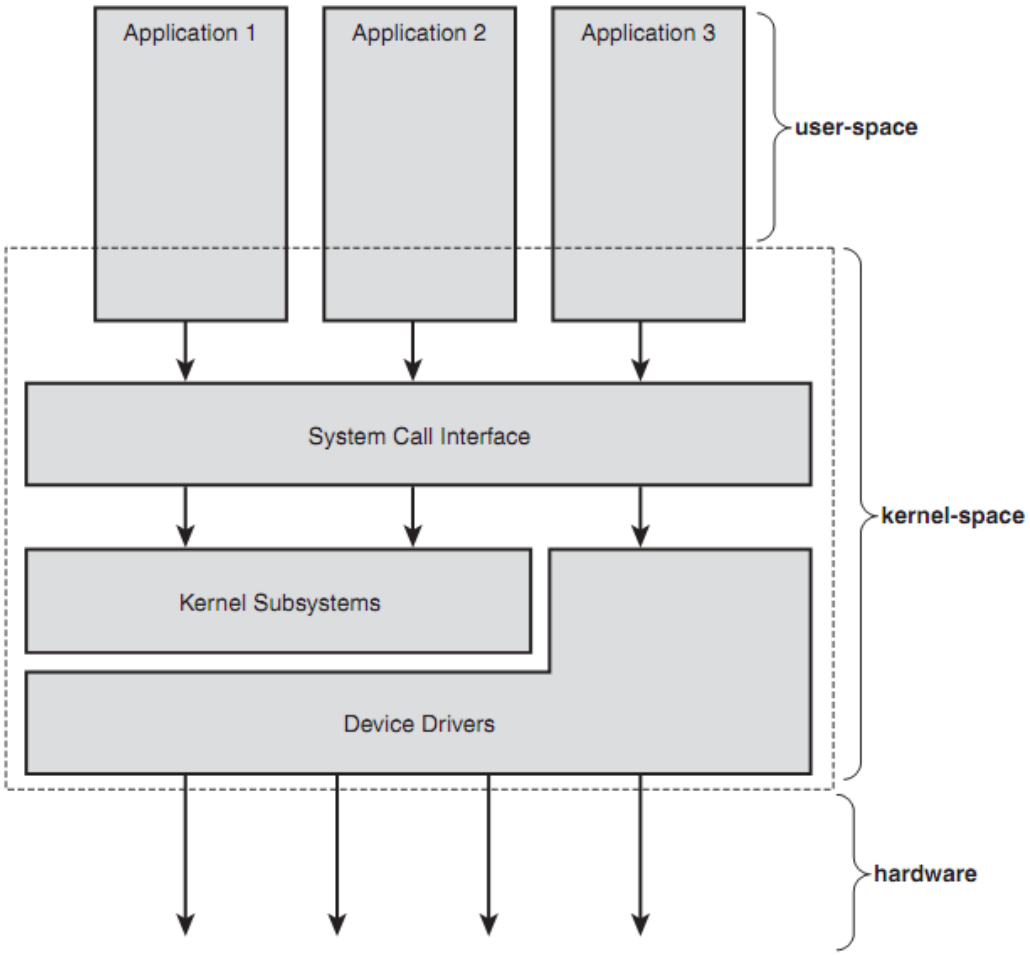

图 1.1 应用、内核和硬件之间的关系

上面描述的这些上下文表示了内核的活动性。实际上在 Linux 中，我们可以把每个进程在任意时刻所做的事情概括为以下三种情况：

- 在用户空间，进程正在执行用户代码。
- 在内核空间的进程上下文，代表特定进程执行。
- 有内核空间的中断上下文，不与任何进程关联，正在处理中断。

这份列表囊括了所有可能的情况，即使是最边角的情况也能适用。例如空闲时，内核正在进程上下文中执行 idle 进程。

Linux 内核对比经典的 Unix 内核

由于共同的祖先和相同的 API，现代 Unix 内核都共享许多设计特性。（请看参考书目中我最喜欢的关于经典 Unix 内核设计的书籍）。通常 Unix 内核都是单一静态的二进制文件，只有极少数例外。也就是说内核是一个单一的、大型的、可执行的镜像，并且运行在单一的地址空间中。Unix 系统通常需要分页内存管理单元（MMU）的支持。MMU 允许系统执行内存保护，为每个进程提供唯一的虚拟地址空间。Linux 历史上也需要 MMU，但特定版本实际上可以脱离 MMU 运行。这是一个很好的特性，允许 Linux 运行在非常小的嵌入式无 MMU 的系统中。不过今天的实际情况是，即使最简单的嵌入式系统也都拥有很多高级特性，包括内存管理单元。因此在本书，我们只关注基于 MMU 的系统。

单内核 VS 微内核设计

我们可以把内核设计分为两个主要的阵营：单内核和微内核（还有一个第三阵营 exokernel，主要用在研究系统中）。

单内核是两个之中更为简单的设计，直到 1980 年代之前所有内核都是如此设计的。单内核完全实现为单一进程，并运行在单一的地址空间中。因此这种内核在硬盘中一般也是单一的静态二进制文件。所有的内核服务都存在并运行于一个大的内核地址空间中。内核内部的通讯非常简单，因为所有的东西都以内核模式运行在相同的地址空间中：内核可以像用户空间应用一样直接调用函数。单内核的优点在于简单和性能。大多数 Unix 系统都设计为单内核。

相反，微内核就不是实现为大的单一进程。内核的功能被分解为许多独立的进程，通常称为 server。理想情况下，只有那些需要特权的 Server 才会运行在特权模式，其它 server 则运行在用户空间。所有 server 都被分开在不同的地址空间，因此直接函数调用是不可能的。微内核通过消息传递来进行交互：消息传递是系统内建的一种进程间通讯进制，不同的 server 通过 IPC 机制发送消息来调用其它 server 的“服务”。不同 server 的分离防止了一个 server 出错导致其它 server 崩溃的情况。同样，系统的模块化也允许 server 的自由替换。

由于 IPC 机制比简单的函数调用开销更大，也由于经常需要在内核空间和用户空间之间互相切换，消息传递会有一定的延迟，吞吐量也不如单内核。因此所有实际的微内核系统现在都把多数或全部 server 放在内核空间中，以消除频繁的上下文切换开销，并允许直接函数调用。Windows NT 内核（Windows XP、Vista 和 Win 7 的基础）和 Mach（Mac OS X 的基础）都是微内核的例子。无论是 Windows NT 还是 Mac OS X，都没有把它们任何的微内核 server 放在用户空间，完全放弃了微内核的主要设计目标。

Linux 是单内核的，也就是说 Linux 内核运行在单一的地址空间，完全在内核模式下执行。但是 Linux 借用了微内核很多优秀的设计原则：拥有模块化设计、抢占自己的能力（称为内核抢占）、支持内核线程、拥有动态加载不同模块的能力（内核模块）。因此 Linux 没有微内核设计的性能问题：所有东西都运行在内核模式，使用直接函数调用而不是消息传递。但是 Linux 却又是模块化的、线程的、内核本身也可以被调度。实用主义再次获胜。

Linus 和其它内核开发者

第 2 章 内核入门