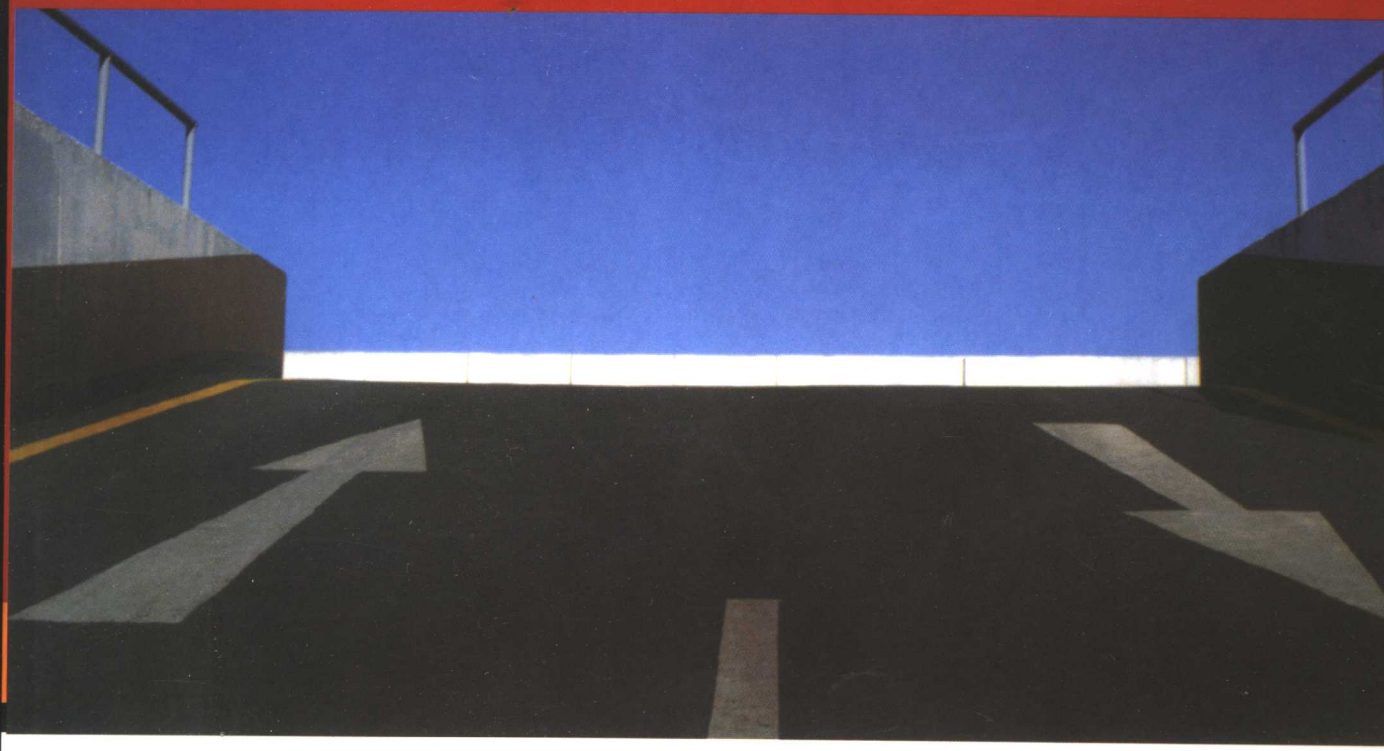
HZ BOOKS

第2版

Linux内核设计与实现

Linux Kernel Development

Second Edition



Novell

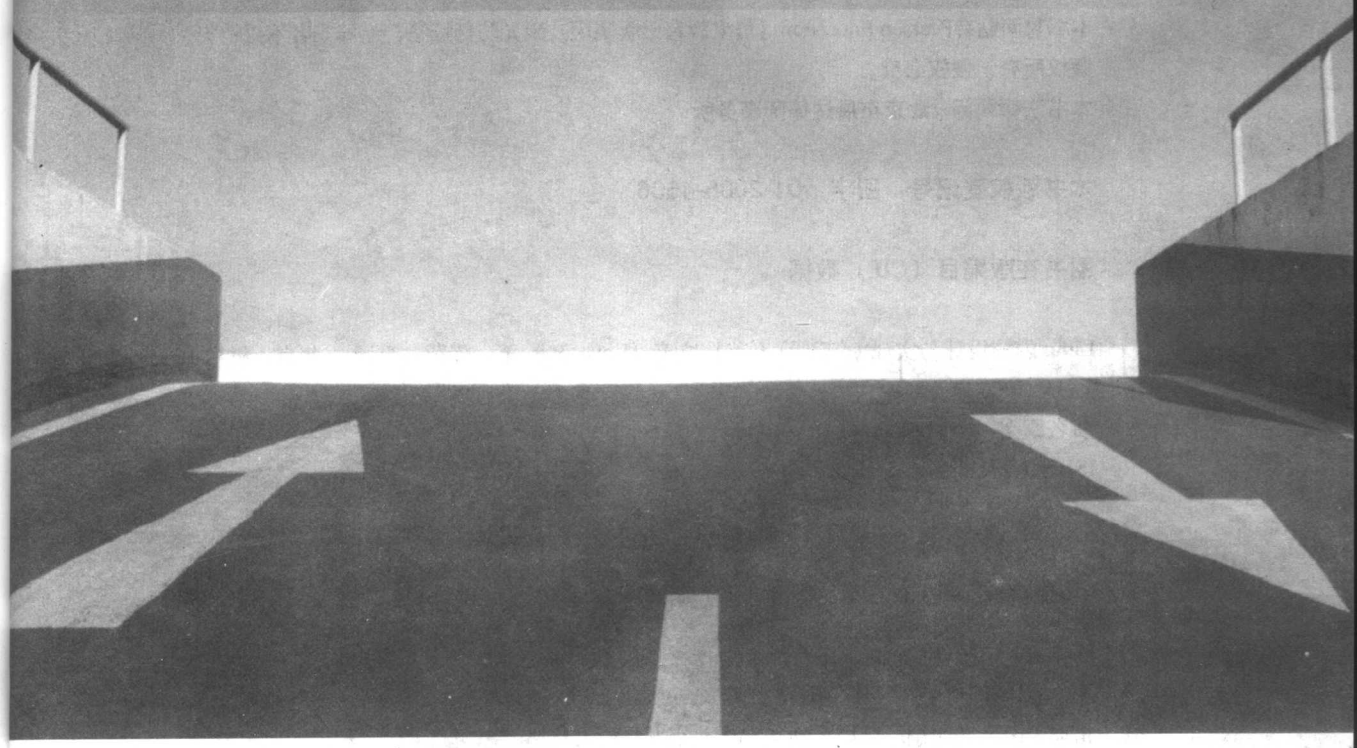
（美） Robert Love 著
陈莉君 康华 张波 译

机械工业出版社
China Machine Press

第2版

Linux内核设计与实现

Linux Kernel Development
Second Edition



(美) Robert Love 著
陈莉君 康华 张波 译

机械工业出版社
China Machine Press

本书基于Linux 2.6内核系列详细介绍Linux内核系统，覆盖了从核心内核系统的应用到内核设计与实现等各方面内容。主要内容包括：进程管理、系统调用、中断和中断处理程序、内核同步、时间管理、内存管理、地址空间、调试技术等。本书理论联系实践，既介绍理论也讨论具体应用，能够带领读者快速走进Linux内核世界，真正开发内核代码。

本书适合作为高等院校操作系统课程的教材或参考书，也可供相关技术人员参考。

本书版权登记号：图字：01-2005-4506

图书在版编目（CIP）数据

Linux内核设计与实现（第2版）/（美）拉芙（Love, R.）著；陈莉君等译. -北京：机械工业出版社，2006.1

书名原文：Linux Kernel Development, Second Edition

ISBN 7-111-17865-3

Ⅰ. L… Ⅱ. ①拉… ②陈… Ⅲ. Linux操作系统-程序设计 Ⅳ. TP316.89

中国版本图书馆CIP数据核字（2005）第133821号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：隋 曦 吴 怡

北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行

2006年1月第2版第1次印刷

787mm×1020mm 1/16 · 19印张

印数：0 001-4 000 册

定价：38.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

译 者 序

不知不觉涉足Linux内核已经几个年头了，与其他有志（兴趣）于此的朋友一样，我们也经历了学习——实用——追踪——再学习的过程。也就是说，我们也是从漫无边际到茫然无措，再到初窥门径，转而觉得心有戚戚焉这一路走下来的。其中甘苦，犹然在心。

Linux最为人称道的莫过于它的自由精神，所有源代码唾手可得。侯捷先生云："源码在前，了无秘密"。是的，但是我们在面对它的时候，为什么却总是因为这种规模和层面所造就的陡峭学习曲线陷入困顿呢？很多朋友就此倒下，纵然Linux世界繁花似锦，纵然内核天空无边广阔。但是，眼前的迷雾重重，心中的阴霾又怎能被阳光驱散呢？纵有雄心壮志，拔剑四顾心茫然，脚下路在何方？

Linux内核入门是不容易，它之所以难学，在于庞大的规模和复杂的层面。规模一大，就不易现出本来面目，浑然一体，自然不容易找到着手之处；层面一多，就会让人眼花缭乱，盘根错节，怎能让人提纲挈领？

"如果有这样一本书，既能提纲挈领，为我理顺思绪，指引方向，同时又能照顾小节，阐述细微，帮助我们更好更快地理解STL源码，那该有多好。"孟岩先生如此说，虽然针对的是C++，但道出的也是研习源码的人们共同的心声。然而，Linux源码研究的方法却不大相同。这还是由于规模和层面决定的。比如说，在语言学习中，我们可以采取小步快跑的方法，通过一个个小程序和小尝试，就可以取得渐进的成果，就能从新技术中有所收获；而掌握Linux呢？如果没有对整体的把握，即使你对某个局部的算法、技术或是代码再熟悉，也无法将其融入实用。其实，像内核这样的大规模的软件，正是编程技术施展身手的舞台（当然，目前的内核虽然包含了一些面向对象思想，但还不能让C++一展身手）。

那么，我们能不能做出点什么，让Linux的内核学习过程更符合程序员的习惯呢？

Robert Love回答了这个问题。Robert Love是一个狂热的内核爱好者，所以他的想法自然贴近程序员。是的，我们注定要在对所有核心的子系统有了全面认识之后，才能开始自己的实践，但却完全可以舍弃细枝末节，将行李压到最小，自然可以轻装快走，迅速进入动手阶段。

因此，本书相对于Daniel P. Bovet和Marco Cesati的内核巨著《Understanding the Linux Kernel》，少了五分细节；相对于实践经典《Linux Device Drivers》，又多了五分说理。可以说，本书填补了Linux内核理论和实践之间的鸿沟，真可谓"一桥飞架南北，天堑变通途"。

就我们的经验，内核初学者（不是编程初学者）可以从这本书着手，对内核各个核心子系统有个整体把握，包括它们提供什么样的服务，为什么要提供这样的服务，又是怎样实现的。而且，本书还包含了Linux内核开发者在开发时需要用到的很多信息，包括调试技术、编程风格、注意事项等等。在消化这本书的基础上，如果你侧重于了解内核，可以进一步研究《Understanding the Linux Kernel》和源代码本身；如果你侧重于实际编程，可以研读《Linux Device Drivers》，直接开始动手工作；如果你想有一个轻松的内核学习和实践环节，请访问我们的网站www.kerneltravel.net。

依然记得译第1版时的喜悦，第2版的到来自然就爱不释手了。同事贺炎为两版之间差异所费的心思全部体现在了字里行间，请读者欣赏第2版丰富的内容吧。

序　言

随着Linux内核和Linux应用程序越来越成熟，越来越多的系统软件工程师涉足Linux开发和维护领域。他们中有些人纯粹是出于个人爱好，有些人是为Linux公司工作，有些是为硬件厂商做开发，还有一些是为内部项目工作的。

但是所有人都必须直面一个问题：内核的学习曲线变得越来越长，也越来越陡峭。系统规模不断扩大，复杂程度不断提高。长此以往，虽然现在的内核开发者对内核的掌握越发炉火纯青，但却会造成新手无法跟上内核发展步伐，出现青黄不接的断层。

我认为这种新老鸿沟已经成为内核质量的一个隐患，而且问题将继续恶化。所以那些真正关心内核的人已经开始致力于扩大内核开发群体。

解决上述问题的一个方法是尽量保证代码简洁：接口定义合理，代码风格一致，“一次做一件事，做到完美”等等。这也就是Linus Torvalds倡导的解决办法。

我提倡的解决办法是对代码慷慨地加上注释：能够让读者立刻了解代码开发者意图的文字（识别意图和实现之间差异的工作称为调试。如果意图不明确显然调试就难以进行）。

可是，即使有注解，也没办法清楚地展现内核的各个主要子系统的全景，说明它们到底要做什么。那么，这些开发者又该从何下手呢？

由文字材料来说明这些在起步阶段就该理解的材料，其实是最合适的。

Robert Love的贡献就在于此，有经验的开发者可以通过本书全面了解内核子系统提供的服务，同时还可以了解这些服务是怎么实现的。对不少人来说，这些知识就已经足够了：那些好奇的人，那些应用程序开发者，那些想对内核的设计品头论足一番的人，都有足够的谈资了。

但是本书同样可以为那些有抱负的内核开发者更上一层楼提供契机，可以帮他们更改内核代码以达到预定的目标。我建议有抱负的开发者能够亲身实践：理解内核某部分的捷径就是对它做些修改，这样能为开发者揭示仅仅通过看内核代码无法看到的深层机理。

严谨认真的内核开发者都应该加入开发邮件列表，不断和其他开发者交流。这是内核开发者相互切磋和并肩前进的最好方法。Robert在本书中对内核生活中至关重要的文化和技巧都做了精彩介绍。

请学习和欣赏Robert的书吧。想必你也希望能精益求精，继续探索，成为内核开发社区中的一员，那么首先你要清楚的是：社区欢迎你。我们评价和衡量一个人是根据他所做的贡献，当你投身于Linux时，你要明白：虽然你仅仅贡献了一小份力，但马上就会有数千万或上亿人受益。这是我们的欢乐之源，也是我们的责任之本。

Andrew Morton
Open Source Development Labs

前　言

在我刚开始有把自己的内核开发经验集结成册，撰写一本书的念头时，我其实也觉得有点头绪繁多，不知道该从何下手。我实在不想落入传统内核书籍的窠臼，照猫画虎地再写这么一本。不错，前人著述备矣，但我终归是要写出点儿与众不同的东西来，我的书该如何定位，说实话，这确实让人颇费思量。

后来，灵感终于浮现出来，我意识到自己可以从一个全新的视角看待这个主题。开发内核是我的工作，开发内核也是我的嗜好，内核就是我的挚爱。这些年来，我不断搜集与内核有关的奇闻轶事，不断积攒关键的开发诀窍，依靠这些日积月累的材料，我可以写一本关于开发内核该做什么——更重要的是——不该做什么的书籍。本质上，这本书仍旧是描述Linux内核是如何设计和实现的，但是写法却另辟蹊径，所提供的信息更倾向于实用。通过本书，你就可以做一些内核开发的工作了——并且是使用正确的方法去做。我是一个注重实效的人，因此，这是一本实践的书，它应当有趣、易读且有用。

我希望读者可以从这本书中领略到更多Linux内核的精妙之处（写出来的和没写出来的），也希望读者敢于从阅读本书和读内核代码开始跨越到开始尝试开发可用、可靠且清晰的内核代码。当然如果你仅仅是兴致所至，读书自娱，那也希望你能从中找到乐趣。

从第1版到现在，又过了一段时间，我们再次回到本书，修补遗憾。本版比第1版内容更丰富：修订、补充并增加新的内容和章节，使其更加完善。第1版之后内核的变化已基本稳定。更值得一提的是，Linux内核联盟做出决定[⊖]，近期内不进行2.7版内核的开发。于是，内核开发者打算继续开发并稳定2.6版。这很有意义，而本书从中的最大受益就是在2.6版内核上可以稳定相当长的时间。内核的相对稳定为捕获其本质留下余地。一本书能最终获得认可，并成为内核的规范文档，是我的希望。

不管你学习Linux的目的是什么，我都衷心地希望你能喜欢我的书。

作者的体会

开发Linux内核不需要天赋，不需要有什么魔法，连Unix开发者普遍长着的络腮胡子都不一定要有。内核虽然有一些有趣并且独特的规则和要求，但是它和其他大型软件项目相比，并没有太大差别。像所有的大型软件开发一样，要学的东西确实不少，但是内核开发并不神秘，也不深奥，Linux内核开发者其实并不需要付出比其他开发者更多的努力。

认真阅读源码非常必要，Linux系统代码的开放性其实是弥足珍贵的，不要无动于衷地将它搁置一边，从而浪费了大好资源。实际上就是读了代码还远远不够呢，你应该钻研并尝试着动手改动一些代码。寻找一个bug然后去修改它，改进你的硬件设备的驱动程序，总之要有的放矢地做一

⊖ 这一决定是在加拿大渥太华2004年夏季举办的Linux内核年度开发者大会上做出的。

些实际工作！只有动手写代码才能真正融会贯通。

内核版本

本书基于Linux 2.6内核系列，具体地说，是基于最新版的2.6.10。内核总在不断更新，一本书很难捕获其动态变化。不过，观其变而抓其质，才是我努力的方向。本书力图呈现着眼于未来的资料，尽可能提供其广泛应用的素材。

读者范围

本书是写给那些有志于理解Linux内核的软件开发者的。本书并不逐行逐字地注解内核源代码，也不是指导开发驱动程序或是内核API的参考手册（如果存在标准的内核API的话）。本书的初衷是提供足够多的关于Linux内核设计和实现的信息，希望读过本书的程序员能够拥有较为完备的知识，可以真正开始开发内核代码。无论开发内核是为了兴趣还是为了赚钱，我都希望能够带领读者快速走进Linux内核世界。本书不但介绍了理论而且也讨论了具体应用，可以满足不同读者的不同需要。全书无处不理论联系实践，也并非一味强调理论或是实践。无论你研究Linux内核的动机是什么，我都希望这本书能将内核的设计和实现分析清楚，起到抛砖引玉的作用。

因此，本书覆盖了从核心内核系统的应用到内核设计与实现等各方面内容。我认为这点很重要，值得花功夫讨论。例如，第7章讨论的是下半部机制。其中分别讨论了内核下半部机制的设计和实现（核心内核开发者会感兴趣），随即介绍了如何使用内核提供的接口实现你自己的下半部（这对设备驱动开发者很有用处）。其实，我认为上述两部分内容是相得益彰的，虽然核心内核开发者主要关注的问题是内核内部如何工作，但是也应该清楚如何使用接口；同样，如果设备驱动开发者了解了接口背后的实现机制，自然也会受益匪浅。

这好比学习某些库的API函数与研究该库的具体实现。初看，好像应用程序开发者仅仅需要理解API——我们被灌输的思想是，应该像看待黑盒子一样看待接口。而另一方面，库的开发者也只关心库的设计与实现。但是我认为双方都应该花时间相互学习。能深刻了解操作系统本质的应用程序开发者无疑可以更好地利用它。同样，库开发者也决不应该脱离基于此库的应用程序，埋头开发。因此，我既讨论了内核子系统的设计，也讨论了它的用法，希望本书能对核心开发者和应用开发者都有用。

我假设读者已经掌握了C语言，而且对Linux比较熟悉。如果读者还具有与操作系统设计相关的经验和其他计算机科学的概念就更好不过了。当然，我也会尽可能多地解释这些概念，但如果你仍然不能理解这些知识的话，请看本书最后参考资料中给出的一些关于操作系统设计方面的经典书籍。

本书很适合在大学中作为介绍操作系统的辅助教材，与介绍操作系统理论的书相搭配。对于大学高年级课程或者研究生课程来说，可直接使用本书作为教材。

本书的相关网站

我维护了一个包含本书相关信息的网站：http://tech9.net/rml/kernel_book/。其中包括本书的勘误表、内容扩展和修改，同时也提供了未来重印和再版的信息。希望读者多到这个站点看看。我也对曾在句尾所加的介词深表歉意，那有点画蛇添足，使句子晦涩难懂，扰乱了读者的视线。

再版感谢

与其他作者一样，我决非是一个人躲在山洞里孤苦地写出这本书来的（那也是一件美差，因为有熊相伴）。我能最终完成本书原稿是与无数建议和关怀分不开的。一页纸无法容纳我的感激，但我还是要衷心地感谢所有给予我鼓励、给予我知识和给予我灵感的朋友和同事。

首先我要对为此书付出辛勤劳作的所有编辑表示感谢，尤其要感谢我的责任编辑Scott Meyers，他为这版的出版自始至终倾其心血。有幸与制作编辑George Nedeff再次愉快合作，他的有条不紊使一切变得顺利。还要特别感谢我的文字编辑Margo Catts，衷心希望我们能像她驾驭文字那样驾驭内核。

本版的技术编辑Adam Belay、Martin Pool和 Chris Rivera也是我需要倍加感谢的人，他们独到的洞察力和准确地校对使书稿质量大大提高。他们的工作称得上完美，如果书中仍留有错误，那是我的责任。忘不了杰出的技术编辑Zack Brown，他对第1版所做的一切依然记忆犹新。

许多内核开发者为我提供了大力支持，回答了许多问题，还有那些撰写代码的人。本书正是由于有了这些代码，才有了存在的意义。他们是：Andrea Arcangeli、Alan Cox、Greg Kroah-Hartman、Daniel Phillips、Dave Miller、Patrick Mochel、Andrew Morton、Zwane Mwaikambo、Nick Piggin和Linus Torvalds。还要特别感谢内核的策划者。

还有许多人在我写作期间不断鼓励我，在方方面面都给予我关怀。这本书也凝聚着他们的爱心。他们是：Paul Amici、Scott Anderson、Mike Babbitt、Keith Barbag、Dave Camp、Dave Eggers、Richard Erickson、Nat Friedman、Dustin Hall、Joyce Hawkins、Miguel de Icaza、Jimmy Krehl、Patrick LeClair、Doris Love、Jonathan Love、Linda Love、Randy O'Dowd、Sal Ribaudo和他了不起的妈妈、Chris Rivera、Joey Shaw、Jon Stewart、Jeremy VanDoren及其家人、Luis Villa、Steve Weisberg 和他的全家、Helen Whisnant等。

最后，我要感谢我的父母，感谢他们的爱。

内核黑客万岁！

Robert Love
剑桥，
马萨诸塞州

◇ ◇ ◇ ◇ ◇

关于作者

Robert Love很早就开始使用Linux，而且一直活跃于开源社区。Robert还是Linux内核和GNOME社团的积极分子。最近，他作为高级内核工程师，受聘于Novell公司参与Ximian桌面组的开发。在此之前，他作为内核工程师受聘于MontaVista软件公司。

他的内核项目包括抢占式内核、进程调度程序、内核事件层，还有VM增强和多任务处理性能

优化。他创建和维护的另外两个开源项目是schedutils和GNOME卷管理器。

除此之外，他对内核有不少精彩评论，而且他还是*Linux Journal*杂志的特邀编辑。

Robert拥有佛罗里达大学数学专业和计算机科学专业学士学位。他出生于佛罗里达南部，目前居住在马萨诸塞州东部的剑桥市。他爱好足球、摄影及烹饪。

◇ ◇ ◇ ◇ ◇

读者反馈方式

本书的所有读者都是我们尊敬的批评者和评论者，我们渴望获得你的意见，我们希望了解我们所做的哪些工作是正确的，哪些工作可以做得更好，你们希望我们出版什么内容的书，以及任何对我们有帮助的建议。

你可以直接发Email或写信给我，告诉我这本书哪里你喜欢，哪里你不喜欢——我们如何提高书的质量。

请注意我无法回答你们的关于书中主题的相关技术问题，我接收的邮件太多，我难以一一回复。请在写信时注明书的作者和书中的主题，同时也请写清楚你的姓名、电话和Email地址。我一定会详细地阅读你的评论，并且与书的作者和编辑交流。

Email：feedback@novellpress.com

通信地址：Mark Taber
Associate Publisher
Novell Press/Pearson Education
800 East 96th Street
Indianapolis, IN 46240 USA

如果希望了解该书更多信息和Novell出版社的其他图书，请访问我们的网站www.novellpress.com。可以在搜索框中输入ISBN（除去数字间的连字符）或书的标题来寻找你要找的书。

目　录

第①章

Linux内核简介

Unix虽然已经使用了30年，但仍然是现存操作系统中最强大和优秀的系统之一。从1969年诞生以来，由Dennis Ritchie和Ken Thompson灵感火花点亮的这个Unix产物已经成为了一种传奇，它历经了时间的考验依然声名始终不坠。

Unix是从贝尔试验室的一个失败的多用户操作系统Multics中涅磐而生的。Multics项目被终止后，贝尔实验室计算科学研究中心的人们处于一个没有交互式操作系统可用的境地。在这种情况下，1969年的夏天，贝尔实验室的程序员们设计了一个文件系统原型，而这个原型最终发展演化成了Unix。Thompson首先在一台无人问津的PDP-7型机上实现了这个全新的操作系统。1971年，Unix被移植到了PDP-11型机中。1973年，整个Unix操作系统被用C语言重写，正是当时这个并不太引人注目的举动，给后来Unix系统的广泛移植铺平了道路。第一个在贝尔实验室以外被广泛使用的Unix版本是第六版，被称为V6。

许多其他的公司也把Unix移植到新的机型上去。伴随着这些移植，开发者们按照自己的方式不断地增强系统的功能，并由此产生了若干变体。1977年，贝尔实验室综合各种变体推出了Unix System Ⅲ。在1983年AT&T推出了 System V[⊖]。

由于Unix系统设计简洁并且在发布时提供源代码，所以许多其他组织和团体都对它进行了进一步的开发。加州大学伯克利分校是其中影响最大的一个。他们推出的变体叫Berkeley Software Distributions (BSD)。伯克利最早的Unix是1979年的3BSD。在3BSD以后，又陆续推出了一系列4BSD系统，包括4.0BSD、4.1BSD、4.2BSD和4.3BSD。这些变体中加入了虚拟内存、换页机制和TCP/IP网络协议栈。1993年，伯克利推出了最后一个官方Unix版本：4.4BSD。他们在这个版本中重写了虚拟内存（VM）。今天，BSD的开发者们仍然在Darwin、Dragonfly BSD、FreeBSD、NetBSD 和OpenBSD中继续他们的梦想。

上世纪80和90年代，许多工作站和服务器厂商推出了他们自己的Unix，这些Unix大部分是在AT&T或伯克利发行版的基础上加上一些满足他们特定体系结构需要的特性。这其中就包括Digital的Tru64、HP的HP-UX、IBM的AIX、Sequent的DYNIX/ptx、SGI的IRIX和Sun的Solaris。

由于最初一流的设计和以后多年的创新和逐步提高，Unix系统成为了一个强大、健壮和稳定的操作系统。下面的几个特点是使Unix具有如此良好弹性的（如此成功的）原因。首先，Unix很简洁。不像其他动辄提供数千个系统调用并且设计目的不明确的系统，Unix仅仅提供几百个系统调用并且有一个非常明确的设计目的。第二，在Unix中，所有的东西都被当作文件对待[⊖]。这种

⊖ 那么System IV呢？据说是个内部开发版本。

⊖ 嗯，确实不能说得太绝对——可是绝大部分是被当作文件的。更新一点的操作系统，比如Unix的下一代Plan9，基本上已经把所有的东西用文件实现了。

抽象使对数据和对设备的操作是通过一套相同的系统调用界面进行：open()、read()、write()、ioctl()和close()。第三，Unix的内核和相关的系统工具软件是用C语言编写的。正是这个特点赋予了Unix令人惊异的移植能力，并且使广大的开发人员很容易就能接受它。此外，Unix的进程创建非常迅速，并且有一个非常独特的fork()系统调用。最后，Unix提供了一套非常简单但又很稳定的进程间通信元语。快速简洁的进程创建过程使Unix的程序把目标放在一次执行集中完成一个任务上，而简单稳定的进程间通信机制又可以保证这些独立的简单程序可以方便地组合在一起，从而完成复杂的多重任务。

今天，Unix已经发展成为一个支持多任务、多线程、虚拟内存、换页、动态链接和TCP/IP网络的现代操作系统。Unix的不同变体应用在大到数百个CPU的集群，小到嵌入式设备的各种系统上。尽管Unix已经不再被认为是一个实验室项目了，但它仍然伴随着操作系统设计技术的进步而继续成长，人们仍然可以把它作为一个通用的操作系统来使用。

Unix的成功归功于其简洁和一流的设计。它能拥有今天的能力和成就应该归功于Dennis Ritchie、Ken Thompson和其他早期设计人员的决策，同时也要归功于那些永不妥协于成见，从而赋予Unix无穷活力的设计抉择。

1.1 追寻Linus的足迹：Linux简介

1991年，Linus Torvalds 为当时新推出的、使用Intel 80386微处理器的计算机开发了一款全新的操作系统，Linux由此诞生。那时，作为芬兰赫尔辛基大学一名学生的Linus，为不能随心所欲使用强大而自由的Unix系统而苦恼。对Torvalds 而言，使用Microsoft的DOS产品不亚于玩波斯王子游戏。Linus热衷使用Minix，一种教学用的廉价Unix，但是，他不能轻易修改和发布该系统的源代码（由于Minix的授权），也不能对Minix开发者所作的设计轻举妄动，这让他耿耿于怀。

Linus像任何一名生机勃勃的大学生一样决心走出这种困境：开发自己的操作系统。他开始写了一个简单的终端仿真程序，用于把自己的终端连接到本校的大型Unix系统上。他不断改进和完善这个终端仿真程序，不久，Linus手上就有了虽不成熟但五脏俱全的Unix。1991年年底，他在Internet上发布了早期版本。

由于Linux得以广泛使用，因此很快赢得众多用户。而实际上，它成功的重要因素是，Linux很快吸引了很多开发者对其代码进行修改和完善。由于其许可证条款的约定，Linux迅速成为多人的合作开发项目。

到现在，Linux早已羽翼丰满了，它被广泛移植到AMD x86-64、ARM、Compaq Alpha、CRIS、DEC VAX、H8/300、Hitachi SuperH、HP PA-RISC、IBM S/390、Intel IA-64、MIPS、Motorola 68000、PowerPC、SPARC、UltraSPARC和v850等各种体系结构上。它覆盖的领域小到手表，大到超级计算机集群。今天，Linux的商业前景也越来越被看好，不管是新成立的Linux专业公司MontaVista 和Red Hat还是闻名遐迩的计算巨头IBM和 Novell，都提供林林总总的解决方案，从嵌入式系统、桌面环境一直到服务器。

Linux克隆了Unix，但Linux不是Unix。需要说明的是，尽管Linux借鉴了Unix的许多设计并且实现了Unix的API（由Posix标准和其他Single Unix Specification定义的），但Linux没有像其他Unix变种那样直接使用Unix的源代码。必要的时候，它的实现可能和其他各种Unix的实现大相径庭，但它完整地达成了Unix的设计目标并且保证了应用程序编程界面的一致。

Linux是一个非商业化的产品，这是它最让人感兴趣的特征。实际上Linux是一个因特网上的协作开发项目。尽管Linus被认为是Linux之父，并且现在依然是一个内核维护者，但开发工作其实是由一个结构松散的工作组协力完成的。事实上，任何人都可以开发内核。和Linux的大部分应用软件一样，Linux内核也是自由（公开）软件[⊖]。当然，也不是无限自由的。它使用GNU的General Public License(GPL)第二版作为限制条款。这样做的结果是，你可以自由地获取内核代码并随意修改它，但如果你希望发布你修改过的内核，你也得保证让得到你的内核的人同时享有你曾经享受过的所有权利，当然，包括全部的源代码[⊜]。

Linux用途广泛，包含的东西也名目繁多。Linux系统的基础是内核、C库、编译器、工具集和系统的基本工具，如登录程序和shell。Linux系统也支持现代的X Windows系统，这样就可以使用完整的图形用户桌面环境，如GNOME。可以在Linux上使用的商业和自由软件数以千计。在这本书以后的部分，当我使用Linux这个词时，我其实说的是Linux内核。在容易引起混淆的地方，我会具体说明到底我想说的是整个系统还是内核。一般情况下，Linux这个词汇主要还是指内核。

1.2 操作系统和内核简介

由于现行一些商业操作系统日趋庞杂及设计上的缺陷，操作系统这个概念被弄得含混不清。许多用户把他们在显示器屏幕上看到的东西理所当然的认为就是操作系统。通常，当然在本书中也这么认为，操作系统是指在整个系统中负责完成最基本功能和系统管理的那些部分。这些部分应该包括内核、设备驱动程序、启动引导程序、命令行shell或者其他种类的用户界面、基本的文件管理工具和系统工具。这些都是必不可少的东西——别以为只要有浏览器和播放器就行了。系统这个词其实包含了操作系统和所有运行在它之上的应用程序。

当然，本书的“操作系统”是关于内核的。用户界面是操作系统的外在表象，内核才是操作系统的内在核心。系统其他部分必须依靠内核这部分软件提供的服务，像管理硬件设备，分配系统资源等等。内核有时候被称作是超级管理者或者是操作系统核心。通常一个内核由负责响应中断的中断服务程序，负责管理多个进程从而分享处理器时间的调度程序，负责管理进程地址空间的内存管理程序和网络、进程间通信等系统服务程序共同组成。对于提供保护机制的现代系统来说，内核独立于普通应用程序，它一般处于系统态，拥有受保护的内存空间和访问硬件设备的所有权限。这种系统态和被保护起来的内存空间，统称为内核空间。相对的，应用程序在用户空间执行。它们只能看到允许它们使用的部分系统资源，并且不能使用某些特定的系统功能，不能直接访问硬件，还有其他一些使用限制（其实，没有固定的结论，例如，它们不再被使用）。当内核运行的时候，系统以内核态进入内核空间，相反，普通用户程序以用户态进入用户空间。应用程序通过系统调用和内核通信来运行（参见图1-1）。应用程序通常调用库函数——比如说C库函数——再由库函数通过系统调用界面让内核代其完成各种不同任务。许多库函数提供的功能并没有单独的系统调用可以替代，在那些较为复杂的函数中，调用内核的操作通常只是整个工作的一个步骤而已。举个例子，拿printf()函数来说，它提供了数据的缓存和格式化等操作，也只是在执行的末期

⊖ 谁有兴趣了解free VS open的论战，请参见http://www.fsf.org and http://www.opensource.org。

⊜ 如果你没读过GNU GPL，你最好还是先看看它吧。内核代码树中COPYING文件就是它的一份拷贝。你也可以在http://www.fsf.org中找到它。