****

**THE LINUX PROGRAMMING INTERFACE**

**A Linux and UNIX System Programming Handbook**

**MICHAEL KERRISK**

no starch press

San Francisco

*Translated by: Kevin*

***本资料仅供学习所用，请于下载后24小时内删除，否则引起的任何后果均由您自己承担。本书版权归原作者所有，如果您喜欢本书，请购买正版支持作者。***

目录

[前言 7](#_Toc289563229)

[主题 7](#_Toc289563230)

[目标读者 7](#_Toc289563231)

[Linux和UNIX 8](#_Toc289563232)

[使用和组织 8](#_Toc289563233)

[例子程序 9](#_Toc289563234)

[练习 10](#_Toc289563235)

[标准和可移植性 10](#_Toc289563236)

[Linux内核和C库版本 11](#_Toc289563237)

[其它语言使用编程接口 11](#_Toc289563238)

[关于作者 11](#_Toc289563239)

[致谢 11](#_Toc289563240)

[许可 12](#_Toc289563241)

[网站和例子程序源代码 12](#_Toc289563242)

[反馈 12](#_Toc289563243)

[第1章 历史和标准 13](#_Toc289563244)

[1.1 UNIX和C简史 13](#_Toc289563245)

[1.2 Linux简史 16](#_Toc289563246)

[1.2.1 GNU项目 17](#_Toc289563247)

[1.2.2 Linux内核 18](#_Toc289563248)

[1.3 标准化 23](#_Toc289563249)

[1.3.1 C编程语言 23](#_Toc289563250)

[1.3.2 第一个POSIX标准 24](#_Toc289563251)

[1.3.3 X/Open公司和开放组织 25](#_Toc289563252)

[1.3.4 SUSv3和POSIX.1-2001 26](#_Toc289563253)

[1.3.5 SUSv4和POSIX.1-2008 28](#_Toc289563254)

[1.3.6 UNIX标准时间线 29](#_Toc289563255)

[1.3.7 实现标准 30](#_Toc289563256)

[1.3.8 Linux、标准、和Linux标准基础 31](#_Toc289563257)

[1.4 小结 32](#_Toc289563258)

[第2章 基础概念 34](#_Toc289563259)

[2.1 操作系统核心：内核 34](#_Toc289563260)

[第3章 系统编程概念 35](#_Toc289563261)

[第4章 文件I/O：统一的I/O模型 36](#_Toc289563262)

[第5章 文件I/O：更多细节 37](#_Toc289563263)

[第6章 进程 38](#_Toc289563264)

[第7章 内存分配 39](#_Toc289563265)

[第8章 用户和组 40](#_Toc289563266)

[第9章 进程凭证 41](#_Toc289563267)

[第10章 时间 42](#_Toc289563268)

[第11章 系统限制和选项 43](#_Toc289563269)

[第12章 系统和进程信息 44](#_Toc289563270)

[第13章 文件I/O缓冲 45](#_Toc289563271)

[第14章 文件系统 46](#_Toc289563272)

[第15章 文件属性 47](#_Toc289563273)

[第16章 扩展属性 48](#_Toc289563274)

[第17章 访问控制列表 49](#_Toc289563275)

[第18章 目录和链接 50](#_Toc289563276)

[第19章 监控文件事件 51](#_Toc289563277)

[第20章 信号：基础概念 52](#_Toc289563278)

[第21章 信号：信号处理器 53](#_Toc289563279)

[第22章 信号：高级特性 54](#_Toc289563280)

[第23章 定时器和睡眠 55](#_Toc289563281)

[第24章 进程创建 56](#_Toc289563282)

[第25章 进程结束 57](#_Toc289563283)

[第26章 监控子进程 58](#_Toc289563284)

[第27章 程序执行 59](#_Toc289563285)

[第28章 进程创建和程序执行的更多细节 60](#_Toc289563286)

[第29章 线程：介绍 61](#_Toc289563287)

[第30章 线程：同步 62](#_Toc289563288)

[第31章 线程：线程安全和线程存储 63](#_Toc289563289)

[第32章 线程：线程取消 64](#_Toc289563290)

[第33章 线程：更多细节 65](#_Toc289563291)

[第34章 进程组、会话和任务控制 66](#_Toc289563292)

[第35章 进程优先级和调度 67](#_Toc289563293)

[第36章 进程资源 68](#_Toc289563294)

[第37章 Daemon 69](#_Toc289563295)

[第38章 编写安全的特权程序 70](#_Toc289563296)

[第39章 能力 71](#_Toc289563297)

[第40章 登录会计 72](#_Toc289563298)

[第41章 共享库基础 73](#_Toc289563299)

[第42章 共享库高级特性 74](#_Toc289563300)

[第43章 进程间通信简介 75](#_Toc289563301)

[第44章 管道和FIFO 76](#_Toc289563302)

[第45章 System V IPC介绍 77](#_Toc289563303)

[第46章 System V消息队列 78](#_Toc289563304)

[第47章 System V信号量 79](#_Toc289563305)

[第48章 System V共享内存 80](#_Toc289563306)

[第49章 内存映射 81](#_Toc289563307)

[第50章 虚拟内存操作 82](#_Toc289563308)

[第51章 POSIX IPC介绍 83](#_Toc289563309)

[第52章 POSIX消息队列 84](#_Toc289563310)

[第53章 POSIX信号量 85](#_Toc289563311)

[第54章 POSIX共享内存 86](#_Toc289563312)

[第55章 文件锁 87](#_Toc289563313)

[第56章 Sockets：介绍 88](#_Toc289563314)

[第57章 Sockets：UNIX Domain 89](#_Toc289563315)

[第58章 Sockets：TCP/IP网络基础 90](#_Toc289563316)

[第59章 Sockets：Internet Domain 91](#_Toc289563317)

[第60章 Sockets：服务器设计 92](#_Toc289563318)

[第61章 Sockets：高级主题 93](#_Toc289563319)

[第62章 终端 94](#_Toc289563320)

[第63章 可选I/O模型 95](#_Toc289563321)

[第64章 伪终端 96](#_Toc289563322)

[附录A：跟踪系统调用 97](#_Toc289563323)

[附录B：解析命令行参数 98](#_Toc289563324)

[附录C：转换NULL指针 99](#_Toc289563325)

[附录D：内核配置 100](#_Toc289563326)

[附录E：更多信息来源 101](#_Toc289563327)

[附录F：部分习题解答 102](#_Toc289563328)

[参考书目 103](#_Toc289563329)

[索引 104](#_Toc289563330)

# 前言

## 主题

本书描述Linux编程接口——Linux（UNIX操作系统的一种免费实现）提供的系统调用、库函数、和其它底层接口。这些接口被直接或间接地使用在Linux上运行的每个程序中。它们允许应用程序完成各种任务：如文件I/O、创建删除文件和目录、创建新进程、执行程序、设置定时器、本机进程和线程间通信、通过网络连接的不同机器进程间通信等等。这些底层接口有时候也叫做系统编程接口。

尽管本书关注于Linux，但我也非常注意标准和可移植性问题，清晰地区分了Linux特有的接口、多数UNIX实现共有的特性、以及POSIX和Single UNIX Specification标准定义的特性。因此本书也提供了UNIX/POSIX编程接口的详尽描述，能够适用于编写UNIX系统应用或跨平台应用的程序员。

## 目标读者

本书主要面向以下读者：

* 为Linux、UNIX、或者其它遵循POSIX的系统开发应用的程序员和软件设计师；
* 在Linux、UNIX、或其它操作系统之间移植应用的程序员；
* Linux或UNIX系统编程课程的教师和高年级学生；
* 希望深入理解Linux/UNIX编程接口，以及系统软件是如何实现的系统管理员和“高级用户”。

我假设你拥有一定的编程经验，但不要求系统编程经验。我还假设你了解C编程语言，并且知道如何使用shell和常用的Linux或UNIX命令。如果你是Linux/UNIX的新手，你会发现第2章非常有用，我们以程序员的视角来讲述Linux和UNIX的基础概念。

## Linux和UNIX

本书原本可以纯粹地讲解标准UNIX（也就是POSIX）系统编程，因为UNIX和Linux的大多数特性都是相同的。不过虽然编写可移植程序是很好的目标，理解Linux对标准UNIX编程接口的扩展也是非常重要的。理由之一是Linux非常流行；其二是有时候为了性能、或使用标准UNIX没有的功能，我们不得不使用非标准的扩展（所有UNIX实现都提供类似的非标准扩展）。

因此本书在适用于标准UNIX的程序员时，还提供了Linux特定编程特性的详细描述。这些特性包括：

* epoll，获得文件I/O事件通知的机制；
* inotify，监控文件和目录改变的机制；
* 能力，授予进程一组超级用户能力的机制；
* 扩展属性；
* i-node标志；
* clone()系统调用；
* /proc文件系统
* Linux对文件I/O、信号、定时器、线程、共享库、进程间通信、和socket的特殊实现细节。

## 使用和组织

你至少可以按两种方式来使用本书：

* 作为Linux/UNIX编程接口的介绍手册。你可以从头到尾阅读本书。后续章节建立在之前章节的基础之上，我尽量避免依赖后续章节的情况。
* 作为Linux/UNIX编程接口的索引参考手册。详细的索引和频繁的交叉引用，允许你随机地阅读任何主题。

我把本书分为以下几部分：

1. 背景和概念：UNIX、C和Linux的历史；UNIX标准简介（第1章）；以程序员的视角介绍Linux和UNIX的基本概念（第2章）；Linux和UNIX系统编程的基本概念（第3章）。
2. 系统编程接口的基础特性：文件I/O（第4章和第5章）；进程（第6章）；内存分配（第7章）；用户和组（第8章）；进程凭证（第9章）；定时器（第10章）；系统限制和选项（第11章）；获取系统和进程信息（第12章）。
3. 系统编程接口的高级特性：文件I/O缓冲（第13章）；文件系统（第14章）；文件属性（第15章）；扩展属性（第16章）；访问控制列表（第17章）；目录和链接（第18章）；监控文件事件（第19章）；信号（第20章到第22章）；定时器（第23章）。
4. 进程、程序、和线程：进程创建、进程结束、监控子进程、执行程序（第24章到第28章）；POSIX线程（第29章到第33章）。
5. 进程和程序的高级主题：进程组、会话、任务控制（第34章）；进程优先级和调度（第35章）；进程资源（第36章）；daemon（第37章）；编写安全的特权程序（第38章）；能力（第39章）；登录会计（第40章）；共享库（第41章到第42章）。
6. 进程间通信（IPC）：IPC简介（第43章）；管道和FIFO（第44章）；System V IPC——消息队列、信号量、共享内存（第45章到第48章）；内存映射（第49章）；虚拟内存操作（第50章）；POSIX IPC——消息队列、信号量、共享内存（第51章到第54章）；文件锁（第55章）。
7. Socket和网络编程：IPC和socket网络编程（第56章到第61章）。
8. 高级I/O主题：终端（第62章）；可选I/O模型（第63章）；伪终端（第64章）。

## 例子程序

我用短小但完整的例子程序来阐述多数接口的使用方法，这些例子都被设计为很容易就能从命令行体验，来查看不同的系统调用和库函数如何工作。所以本书包含大量的示例代码——大概15000行C代码和shell会话日志。

尽管阅读和试验例子程序是不错的起点，掌握本书讨论的概念最有效的方法是编写代码，按你的想法修改例子程序，或者编写新程序都可以。

本书的所有源代码都可以在网站上下载。源代码包含许多书中没有的程序。这些程序的目的和细节在注释中都有相关描述。我提供了Makefile编译这些程序，以及一个README文件，给出了例子程序更多的细节信息。

源代码采用GNU Affero 通用公共授权版本3，可以自由分发和修改。源代码中也包含一份该协议的拷贝。

## 练习

多数章节都以一组练习结束，其中一些是要你按不同方式来试验例子程序，另外一些是该章讨论过的概念相关的问题，还有就是要求你来编写代码以巩固你对本书的理解。你可以在附录F找到部分练习的解答。

## 标准和可移植性

贯穿整本书，我都对可移植性问题特别地关注。你会发现很多相关标准的引用，特别是POSIX.1-2001和Single UNIX规范版本3（SUSv3）标准。同时你还将看到这些标准最新修订的细节改变，也就是POSIX.1-2008和SUSv4标准。（由于SUSv3是更大的修订版本，也是本书编写时最广泛有效的UNIX标准，本书讨论的标准大多是SUSv3，并标注出SUSv4不同的地方。除非我明确地提到，你可以假设我们对SUSv3规范的描述也适用于SUSv4）。

对于那些不是标准的特性，我会指出在不同UNIX实现间的差别。我还会突出那些Linux特定的特性，以及Linux与其它UNIX对系统调用和库函数实现上的细小差别。当某个特性我没有明确指出是Linux专有时，你也通常可以假设它在多数或所有UNIX上都有实现。

本书大多数例子程序我都在Solaris、FreeBSD、Mac OS X、Tru64 UNIX、和HP-UX上测试通过（除了那些Linux独有的特性）。为了提高代码在这些系统上的可移植性，本书网站上提供的某些例子程序有一些额外的代码。

## Linux内核和C库版本

本书主要关注Linux 2.6.x系列，这是本书写作时最广泛使用的内核版本。Linux 2.4的某些细节也会提到，我也会指出Linux 2.4和2.6的区别。当Linux 2.6.x系列出现了新特性时（例如2.6.34），我也会特别指出相应的内核版本号。

至于C库，本书则主要关注于GNU C库（glibc）版本2。当然，glibc 2.x系列版本存在差异时，我也会特别指出。

在本书即将印刷时，Linux内核刚刚发布了2.6.35版本，glibc则已经发布2.12版本。本书完全适用于这两个软件版本。Linux内核和glibc将来接口的变化，会在本书的网站上列出。

## 其它语言使用编程接口

尽管例子程序用C语言编写，你也可以在其它编程语言中使用本书讨论的接口——例如编译型语言C++、Pascal、Modula、Ada、FORTRAN、D；解释型语言Perl、Python、Ruby等。（Java则需要采用一种不同的方式JNI）。不同的语言要获取必要的常量定义和函数声明，需要使用不同的技术（C++除外），另外传递函数参数时可能也需要一点额外的工作。此外就没有太大的区别了，核心概念其实都是一样的。因此即使你使用其它的编程语言，你也会发现本书提供的信息是适用的。

## 关于作者

（略）

## 致谢

（略）

## 许可

电子工程学会和开放组织非常友好地许可我引用IEEE Std 1003.1, 2004版本，以及信息技术标准——可移植操作系统接口（POSIX），开放组织基本规范Issue6。完整的标准可以在<http://www.unix.org/version3/online.html>上在线查阅。

## 网站和例子程序源代码

你可以在<http://www.man7.org/tlpi>上找到关于本书更多的信息，包括勘误表和例子程序的源代码。

## 反馈

我非常欢迎代码bug报告、代码改进建议、以及代码可移植性的提高。同样我也欢迎本书的bug报告和改进建议。由于Linux编程接口总是在变化，我也非常高兴能获得关于本书将来版本的改进意见，包括新特性和变化特性。

Michael Timothy Kerrisk

Munich, Germany and Christchurch, New Zealand

August 2010

[mtk@man7.org](mailto:mtk@man7.org)

# 第1章 历史和标准

Linux是UNIX操作系统家族的成员之一。在计算机的术语里，UNIX已经拥有很悠久的历史。第1章的前半部分简述UNIX的历史。我们首先描述UNIX系统和C编程语言的起源，然后讲述导致Linux发展成为今天这个样子的两个关键因素：GNU项目和Linux内核的开发。

UNIX系统最显著的特点之一是它的开发不是被一个厂商或组织控制。相反许多商业和非商业组织都为UNIX的发展做出了贡献。UNIX也因此增加了许多革新的特性，但同时也导致UNIX各个实现之间的分歧越来越大，编写一个能运行于所有UNIX实现的应用也变得非常困难。于是产生了UNIX的标准化运动，我们将在本章后半部分进行讨论。

## 1.1 UNIX和C简史

第一个UNIX由贝尔实验室（电话公司AT&T的一个部门）的Ken Thompson在1969年开发完成（Linus Torvalds也正是在这一年出生）。这个UNIX是用汇编为Digital PDP-7微计算机编写。UNIX这个名字和MULTICS（Multiplexed Information and Computing Service）有关，后者是AT&T与麻省理工学院（MIT）和通用电子之前合作开发的操作系统项目。（由于该项目最初的失败，没有能够开发出一个有用的系统，当时AT&T已经退出项目）。Thompson的新操作系统从MULTICS中借用了一些设计，包括树型结构文件系统、对命令解释执行采用独立的程序（shell）、以及把文件当作无结构的字节流。

在1970年， UNIX使用汇编语言为新的Digital PDP-11微计算机重新编写，这个PDP-11的遗留痕迹至今仍然可以在多数UNIX实现中找到，包括Linux。

不久之后，Dennis Ritchie，Thompson在贝尔实验室的一个同事，设计和实现了C编程语言。这是一个进化的过程，C起源于更早的解释语言B，最初由Thompson实现了B语言，并从一个更早的语言BCPL中借鉴了许多想法。到1973年，C已经成熟到UNIX内核几乎可以全部使用其重写。UNIX也因此成为最早使用高级语言编写的操作系统，使其迁移到其它硬件体系架构成为可能的重要因素。

C语言的这个起源，解释了C和C++成为今天最广泛的系统编程语言的原因。之前广泛使用的语言都是为其它目的而设计的：FORTRAN为工程师和科学家完成数学任务；COBOL为商业系统处理面向记录的数据流。C填补了一个空白，和FORTRAN、COBOL不一样的是，C语言是几个人为了一个目标而设计的：开发一个高级语言来实现UNIX内核和相关的软件。和UNIX操作系统本身一样，C由专业的程序员为自身所设计。所产生的语言是小巧、高效、强大、简洁、模块化、注重实效、和一致的。

UNIX第一至第六版

在1969年到1979年间，UNIX发布了一系列版本。本质上就是AT&T对UNIX开发进展的一个快照。UNIX最初的六个版本发布时间如下：

* 第一版，1971年11月：此时UNIX还运行在PDP-11上，已经拥有一个FORTRAN编译器，和许多今天依然在使用的工具，包括ar, cat, chmod, chown, cp, dc, ed, find, ln, ls, mail, mkdir, mv, rm, sh, su, who。
* 第二版，1972年6月：UNIX安装在AT&T内部的10台机器上。
* 第三版，1973年2月：这个版本包含一个C编译器和管道的最初实现。
* 第四版，1973年11月：第一个几乎全部用C编写的版本。
* 第五版，1974年6月：此时UNIX已经安装在超过50个系统中。
* 第六版，1975年5月：这是第一个在AT&T范围外广泛使用的版本。

在这些版本发布的过程中，UNIX的使用和声望得到了扩展，首先在AT&T内部，随后在外部。Communications of the ACM杂志发表的一篇关于UNIX的论文也为此做出了巨大贡献。

当时AT&T正在接受美国电话系统对其垄断的政府制裁。AT&T与美国政府的协议禁止其销售软件，这也意味着AT&T不能把UNIX作为产品销售。相反，从1974年的第五版开始，特别是第六版，AT&T授权大学免费使用UNIX。针对大学的UNIX发布版包含文档和内核源代码（当时大约10000行）。

AT&T对大学发布UNIX极大地促进了UNIX的使用和流行，到1977年UNIX已经运行在500个地方，包括125所美国大学和其它一些国家。当时的商业操作系统非常昂贵，而UNIX为大学提供了一个交互式多用户的操作系统，即便宜又强大。同时UNIX还给大学计算机科学研究提供UNIX操作系统的源代码，他们可以修改并提供给学生学习和体验。很多学生学习了UNIX之后，就成为了UNIX的布道者。其它则加入或组建自己的公司，销售运行着UNIX操作系统的计算机工作站。

**BSD和System V的诞生**

1979年1月UNIX发布了第七版，改进了系统的可靠性，提供了一个增强的文件系统。这个发布版还包含一些新的工具，包括：awk, make, sed, tar, uucp, Bourne shell, 和FORTRAN 77编译器。第七版的发布对于UNIX来说具有重要意义，因为从这一刻起，UNIX产生了两个重要的变种：BSD和System V，它们的起源我们马上就会简要地描述。

Thompson在1975/1976学年回到自己的母校，加州大学伯克利分校担任客座教授。在那里他和几个毕业生为UNIX增加了许多新特性。（其中一个学生Bill Joy，随后与别人一起组建了Sun Microsystems，成为UNIX工作站市场早期参与者）。Berkeley开发了许多新的工具和特性，包括C shell、vi编辑器、改进的文件系统（Berkeley Fast File System）、sendmail、Pascal编译器、新的Digital VAX体系架构下的虚拟内存管理等。

在Berkeley Software Distribution（BSD）的授权许可下，这个版本的UNIX，包括它的源代码，被广泛地发布出去。1979年发布了第一个完整发行版3BSD（更早的Berkeley-BSD和2BSD，只是增加Berkeley开发的新工具，而不是完整的UNIX发行版）。

到1983，加州大学伯克利的计算机系统研究组织（Computer Systems Research Group）发布了4.2BSD。这是一个重大的发行版，因为它包含了完整的TCP/IP实现，包括socket应用编程接口（API）和许多网络工具。4.2BSD和它的前任4.1BSD被广泛发布于全世界的许多大学。它们也构成了Sun公司的UNIX变种，SunOS（1983首次发布）的基础。其它重要的BSD发布包括1986年的4.3BSD，以及1993年的最终发布版：4.4BSD。

与此同时，US反托拉斯诉讼强制AT&T解散（法律诉讼起于1970年代中期，1982年解散生效），由于在电话系统中不再垄断，公司被允许运营UNIX。结果就是1981年System III的诞生。AT&T的UNIX支持组（USG）负责开发System III，它雇佣了数百名开发者来增强UNIX，和开发UNIX应用（著名的有document preparation package和软件开发工具）。随后在1983年发布了System V(5)的第一个版本，一系列的小发布版后最终是1989年的System V发布版4（SVR4），到这时System V已经吸收了BSD的许多特性，包括网络基础设施。System V授权给许多商业厂商，这些厂商使用System V作为自己UNIX实现的基础。

因此到1980年代末，除了各种BSD发布版在大学广泛使用，UNIX还在许多硬件上拥有各种商业实现：包括Sun的SunOS及随后的Solaris、Digital的Ultrix和OSF/1（经过一系列的改名和收购之后，成为了今天的HP Tru64 UNIX）、IBM的AIX、Hewlett-Packard（HP）的HP-UX、NeXT的NeXTStep、Apple Macintosh的A/UX、Microsoft和SCO为Intel x86-32体系架构开发的XENIX。（本书将Linux的x86-32实现统一称为Linux/x86-32）。这种状况和当时典型的私有硬件/操作系统的方式完全不同，后者通常是厂商只生产一个或少数私有计算机芯片体系架构，然后在上面销售自己的私有操作系统。多数厂商系统的这种私有属性，意味着购买受限于一个厂商。切换到另一种私有操作系统和硬件平台会非常昂贵，因为需要迁移现有应用并进行相关的重新训练。这个因素再加上各个厂商便宜的单用户UNIX工作站，使得可移植的UNIX系统对商业应用非常具有吸引力。

## 1.2 Linux简史

Linux这个术语通常引用基于Linux内核的完整的类UNIX操作系统。不过这是错误的叫法，因为典型商业Linux发行版的许多关键组件，都起源于另一个项目，这个项目比Linux要早好几年。

### 1.2.1 GNU项目

Richard Stallman是一个天才程序员，曾工作于MIT，他在1984年开始考虑实现一个"Free" UNIX。Stallman对"free"的观点是精神上的自由，并且定义在法律层面上，而不仅仅是免费（参考<http://www.gnu.org/philosophy/free-sw.html>）。无论如何，Stallman倡导的自由也就意味着软件（如操作系统）应该免费或非常便宜。

Stallman大大影响了厂商对私有操作系统系统附加的限制。这些限制意味着购买计算机软件通常不包含源代码，而且通常不能对该软件进行复制、修改、和分发。Stallman指出这种形式鼓励程序员互相竞争并且保密自己的工作，而不是互相合作和共享成果。

于是Stallman创建了GNU项目（GNU's not UNIX），目标是开发一个完整、自由、类UNIX的系统，包含一个内核和所有相关的软件包，并且鼓励其它人参与该项目。到1985年，Stallman成立了自由软件基金会（FSF），这是一个旨在支持GNU项目以及其它自由软件开发的非赢利组织。

GNU项目的一个重要成果就是GNU General Public License(GPL)的产生，这也是Stallman对自由软件精神的具体化。Linux发行版的多数软件，包括内核都按GPL（或者类似的许可）授权。GPL授权的软件必须使源代码自由可用，而且允许按GPL许可自由地重新发布。GPL授权的软件允许自由地修改，但是修改后的软件必须同样遵循GPL许可。如果修改后的软件以可执行方式发布，作者必须同时允许以不超过发布的代价获得修改过的源代码。GPL第一版发布于1989年，目前的版本3发布于2007年。版本2发布于1991年，目前使用最广泛，也是Linux内核采用的授权。

GNU项目最初并没有开发出一个可用的UNIX内核，但确实创建了许多其它程序。由于这些程序设计成在类UNIX操作系统中运行，它们可以也确实被用在现有的UNIX实现中，有些还迁移到其它操作系统。GNU项目最著名的程序有Emacs文本编辑器、GCC（最早是GNU C编译器，不过现在重新命名为GNU编译器集合，包含C、C++和其它语言的编译器）、Bash shell、和glibc（GNU C库）。

在1990年代初期，GNU项目已经拥有了一个几乎完整的系统，除了一个关键的组成：可用的UNIX内核。GNU项目开始规划一个野心勃勃的内核设计，被称为GNU/HURD，基于Mach微内核。不过HURD远远达不到可发布的程度。（在本书写作之时，HURD的工作仍在继续，目前只能运行在x86-32体系架构下）。

万事俱备，只欠东风。GNU项目已经创建了完整UNIX系统所需的一切，只差一个最重要的内核了。

### 1.2.2 Linux内核

Linus Torvalds在1991年还是芬兰郝尔辛基大学的一名学生，当时他想为自己的Intel 80386 PC编写一个操作系统。在Linus的课程学习过程中，他接触了Minix，由Andrew Tanenbaum在1985年左右开发的类UNIX操作系统内核，后者是荷兰某大学的教授。Tanenbaum创造了Minix，并提供完整的源代码，用作大学操作系统设计课程的教学工具使用。Minix内核可以在386系统中构建和运行，但是由于主要目的是教学工具，Minix设计成很大程度上独立于硬件体系架构，因此不能完全发挥386处理器的能力。

于是Torvalds启动了自己的项目，开始为386创建一个高效、全功能的UNIX内核。几个月之后，Torvalds开发了一个基本的内核，允许自己编译和运行许多GNU程序。然后在1991年10月5日，Trovalds开始在网上请求其它程序员的帮助，发出了下面这段被广泛引用的声明，他在comp.os.minix Usenet新闻组上发布了自己内核的0.02版：

你是否怀念 minix-1.1 版时的日子？那时人们干劲十足，自己编写设备驱动程序。你是否手头正缺少一个很好的项目，并且非常渴望为符合自己的需要动手修改一个操作系统？当几乎所有的程序都能在Minix上运行时，你是否感到非常失望？不再有为了调通一个巧妙的程序而整夜不睡觉的夜猫子？那么本消息（邮件、公告）可能正是为你而发布的:-)。

正如我一个月前所提到的，我正在开发一个用于 AT-386 微机类似于Minix 的操作系统。它目前已经达到了可用的程度(当然，能不能用还依赖于你的具体要求)，而且我很高兴把源代码拿出来广泛发布。目前它的版本是0.02(加上已经编制好的(很小的)补丁程序，就是0.03)，但是我已经在它上面成功地运行了 bash/gcc/gnu-make/gnu-sed/压缩程序等。

该小巧项目的源程序可以在nic.funet.fi(128.214.6.100)上/pub/OS/Linux 目录中找到。该目录中含有一些README文件以及几个在Linux下运行的二进制执行程序(bash, update和gcc，你还能要求什么呢:-)。提供了完整的内核源代码，而且没有使用minix的代码。库文件的源代码仅是部分免费的，所以目前不能给出。照内核现在的样子，系统已经可以进行编译，并且已经可以运行。二进制执行程序（bash和gcc）的源代码可以在同一个地方的/pub/gnu目录中找到。

当心！警告！注意！这些源代码仍然需要minix-386系统来进行编译（需要 gcc-1.40，1.37.1可能也能用，但没有试过），并且如果你想运行它的话还需要minix来进行设置，所以对没有minix的人来说，它至今它还不是一个独立的系统，不过我正在朝这方面努力着。你还需要有些骇客的本事来设置它，所以对那些希望一个minix-386取代品的人来说，就不用考虑Linux了。它目前主要是供对操作系统感兴趣的骇客使用的，并且有能使用minix的386机器。该系统需要一个AT兼容硬盘（IDE硬盘当然更好）以及EGA/VGA显示卡，如果你还感兴趣的话，就使用ftp下载README/RELNOTES文件看看，并且/或者给我EMAIL告之其它信息。

我能够（当然，几乎是）听到你问自己“为什么？”，Hurd将在近年（或者两年、或者下个月，谁知道）内推出，而且我已经有了minix。这是一个骇客为骇客们写的程序，在开发过程中我已经得到了快乐，而某些人可能也乐意阅读它，甚至为自己的需要而修改它。它仍然很小，足以理解、使用和修改，我正期望你可能有的任何建议和说明。 我也对为minix系统编写过工具软件/库函数的任何人的反馈信息感兴趣。如果你的软件是可以自由发布的(在版权下甚至公共域内)，那么我很希望得到你们的消息，这样我就可以将它们加入到Linux系统中。现在我正使用着Earl Chews的stdio（Earl，谢谢你的很好而又能使用的系统），很欢迎这种类似的软件。你的版权当然会保留着，如果你乐意我使用你的代码，就请告知。

Linus

按照传统UNIX克隆采用的X字母结尾命名惯例，这个内核最终命名为Linux。最初Linux采用更加受限制的授权，不过Torvalds很快就将Linux许可更换为GNU GPL协议。

Linus的请求帮助得到热烈影响。很多程序员加入Linux的开发，添加了许多特性，例如增强的文件系统、网络支持、设备驱动、和多处理器支持等。到1994年3月，开发者们发布了1.0版本，1995年3月发布了Linux 1.2，1996年6月发布了Linux 2.0，1999年1月发布了Linux 2.2，2001年1月发布了Linux 2.4。2001年11月开始内核2.5的开发，到2003年12月发布了Linux 2.6。

**BSD**

值得一提的是1990年代前期，另一个免费的UNIX也已经能够用于x86-32体系架构。Bill和Lynne Jolitz对一个已经很成熟的BSD系统向x86-32做了迁移，名叫386/BSD。迁移基于BSD Net/2发布版（1991年6月），是4.3BSD的一个版本，把所有AT&T私有的源代码都替换或移除掉。Jolitz夫妇把Net/2迁移到x86-32，并重写了缺失的代码，在1992年2月发布了386/BSD的第一个版本（V0.0）。

在经历了最初短暂的成功和流行之后，386/BSD的工作由于各种原因而停滞。随着大量patch逐渐积压得不到处理，两个开发团队应运而生，分别创建了自己基于386/BSD的发布版：NetBSD，强调在各种硬件之间保持可移植性；FreeBSD，强调性能，也是现代BSD中最流行的一个。NetBSD的第一个发布版是1993年4月的0.8；FreeBSD的首张CD-ROM（版本1.0）发布于1993年12月。另外还有一个OpenBSD，派生自NetBSD项目，在1996年发布了最初的2.0版本，OpenBSD特别强调安全性。到2003年中期，一个新的DragonFly BSD又从FreeBSD 4.x分离而出。DragonFly BSD采用了不同于FreeBSD 5.x的方式，特别为对称多处理器（SMP）体系架构设计。

如果不提到UNIX系统实验室（USL，负责开发和销售UNIX的AT&T子公司）和伯克利之间的诉讼，那我们对于BSD的讨论就不是完整的。在1992年初，合并成立了伯克利软件设计公司（BSDi，今天是Wind River的一部分），开始发布一个商业支持的BSD UNIX：BSD/OS，基于Net/2发行版和Jolitz夫妇的386/BSD增强功能。BSDi以995美元发布二进制和源代码，并且建议潜在客户使用他们的电话号码1-800-ITS-UNIX。

1992年4月，USL向BSDi正式提出诉讼，试图阻止BSDi销售包含USL私有源代码和商业秘密的产品。USL同时还要求BSDi停止使用迷惑性的电话号码。这个官司最终扩大为要求加州大学赔偿。法院最后判决同意了USL的两个主张，并驳回了其它请求。接着马上加州大学向USL提出反诉讼，声称USL未经许可在System V中使用了BSD代码。

官司正在悬而未决的时候，Novell收购了USL，其CEO（Ray Noorda）开始公开声明自己希望双方在市场上而不是法院里竞争。诉讼最终得以在1994年1月终结，加州大学必须移除Net/2发布版18000个文件中的3个，并对其它少数文件做一些很小的修改，另外还要对大约70个文件增加USL版本声明，而且这些文件不能够再次发布。这个修改后的系统在1994年6月发布为4.4BSD-Lite（加州大学发布的最后一个版本是1995年6月的4.4BSD-Lite版本2）。从这时开始，法律条款要求BSDi、FreeBSD、NetBSD用修改后的4.4BSD-Lite源代码替换Net/2。尽管这导致BSD派生开发的一定延迟，但也使这些系统通过三年的开发，从加州大学计算机系统研究组织发布Net/2后重新同步到一起。

**Linux内核版本号**

和多数自由软件项目一样，Linux采用尽早发布、经常发布的模型，因此新的内核修订频繁更新（有时候几乎每天）。随着Linux用户群的增长，对发布模型进行了一定的修改，以减少对现有用户的影响。具体来说，从Linux 1.0发布之后，内核开发者就采用了固定的内核版本命名规范，每个发布版本统一命名为x.y.z：其中x表示主版本号；y表示在该主版本号下的副版本号；而z则是副版本号下的修订版本号（通常是很小的改进和bug修复）。

在这样一种模型下，通常会有两个内核版本总是处在开发过程中：一个是稳定版，用于生产系统，其主版本号为偶数；另一个是开发版，相对来说不稳定一些，主版本号一般是下一个奇数。理论上（实践中并不总是）所有新特性都只应该添加在当前开发版内核中，而稳定版的修订系列严格限制为很小的改进和bug修复。当内核开发者认为开发版本适合发布时，就会成为新的稳定版，并赋予一个偶数版本号。例如2.3.z开发内核最终形成了2.4稳定内核版本。

2.6内核发布之后，开发模型发生了变化，主要目的是解决稳定版内核发布时间间隔太长导致的问题和挫折（Linux 2.4.0和2.6.0之间差不多有三年时间）。关于改善开发模型的谈论时不时都有进行，但是核心细节基本保持如下：

* 不再有稳定和开发版的明确区分。每个新的2.6.z发布都可以包含新特性，而且都经历增加新特性，然后通过几个候选发布版达到稳定的生命周期。当候选版本足够稳定时，就发布为内核2.6.z版本。发布周期大约三个月。
* 有时候稳定的2.6.z发布版需要小的patch来修复bug或安全性问题。如果这些修复有足够高的优先级，而且这些patch也足够简单到不可能出错，那么不需要等待下一个2.6.z发布版，可以直接创建一个2.6.z.r发布版，这里的r序列号表示2.6.z内核的副修订版本。
* 额外的责任被转移到发行版厂商，来确保发行版内核的稳定性。

后面章节有时候遇到特殊的API时，会提及具体的内核版本（例如新的或修改的系统调用）。不过在2.6.z系列内核之前，多数内核变更都发生在奇数开发版中，我们通常会注明这个变化是在下一个稳定版中产生的，因为多数应用开发者都是使用稳定版内核而不是开发版内核。许多情况下，手册页则会精确地标注某个特性是在哪个开发版出现或修改的。

对于2.6.z系列内核出现的变化，我们会标注具体的内核版本号。当我们说某个特性是内核2.6的新特性时，如果不带z修订号，就表示这个特性是在2.5开发内核中实现的，首次出现在稳定内核版本2.6.0。

**移植到其它硬件体系架构**

在Linux最初的开发阶段，高效地实现Intel 80386是主要目标，而不是与其它处理器体系架构的可移植性。但是随着Linux越来越流行，开始向其它处理器体系架构进行移植，最开始是Digital Alpha芯片。Linux能够支持的硬件体系架构非常多，而且还在不断增长。包括但不限于：x86-64、Motorola/IBM PowerPC和PowerPC64、Sun SPARC和SPARC64（UltraSPARC）、MIPS、ARM（Acorn）、IBM z系列（以前的System/390）、Intel IA-64（Itanium）、Hitachi SuperH、HP PA-RISC、和Motorola 68000。

**Linux发行版**

准确地说，Linux这个术语只是指Linus Torvalds和其它开发者开发的内核。但是通常我们说的Linux则包括内核，加上大量其它软件（工具和库），它们一起组成了完整的操作系统。在Linux最早期的时代，用户需要自己组合所有这些软件，创建文件系统，正确地存放和配置文件系统中的所有软件。这需要大量时间和专业知识。结果就是Linux发行版市场的兴起，发行版自动化处理大多数安装过程，创建文件系统并安装内核和其它必需的软件。

最早的发行版出现于1992年，包含了MCC Interim Linux（Manchester Computing Centre，UK）、TAMU（Texas A&M 大学）、和SLS（SoftLanding Linux系统）。现存最老的商业发行版是1993年出现的Slackware；非商业的Debian发行版大约也在那时候出现，随后是SUSE和红帽。当前非常流行的Ubuntu发行版于2004年发布。今天许多发行版公司都雇佣了大量程序员，继续为自由软件项目做出贡献，或者发起新的项目。

## 1.3 标准化

1980年代后期，众多的UNIX实现也带来一个问题。某些UNIX实现基于BSD，其它则基于System V，某些特性则同时来自这两个变种。此外每个商业厂商都为自己的UNIX实现增加了额外的特性。结果就是从一个UNIX实现向另一个移植软件变得非常困难。这种状况为C编程语言和UNIX系统的标准化施加了积极的压力，标准化可以使应用在平台间移植就得非常简单。我们来看一看相关的标准。

### 1.3.1 C编程语言

在1980年代早期，C已经存在了10年之久，而且在多数UNIX系统和其它操作系统中都已经被实现。各种不同实现之间存在许多细小的差别，部分原因是C语言某些方面如何工作，并没有在事实上的标准中（Kernighan和Ritchie在1978年出版的C编程语言一书）详细描述（书中老式的C语法有时候也称为传统C或者K&R C）。此外，1985年产生的C++突出了C语言中缺乏的某些不影响兼容性的改进或增强，比如函数原型、结构体赋值、类型限定符（const和volatile）、枚举类型、和void关键字。

这些因素驱动了C语言的标准化，最终在1989年通过了美国国家标准协会（ANSI）的C标准（X3.159-1989），随后又在1990年被采纳为国际标准组织（ISO）标准（ISO/IEC 9899:1990）。除了定义C语言的语法和语义，该标准还描述了标准C库，包括stdio函数、字符串处理函数、数学函数、各种头文件等等。这个版本的C被称为C89或ISO C90，Kernighan和Ritchie的C编程语言第二版（1988）对标准做了完整描述。

ISO在1999年接受了C标准的新修订（ISO/IEC 9899:1999；参考<http://www.open-std.org/jtc1/sc22/wg14/www/standards>）。这个标准通常称为C99，对C语言和标准库做了一定的修改。包括增加long long和bool数据类型、C++风格注释（//）、受限指针、以及变量长度数组。（在本书写作的时候，还在对C标准进行进一步的修订，非正式地命名为C1X。新标准有望在2011年获得批准）。

C标准与操作系统实现完全无关；也就是说并没有绑定于UNIX系统。这表示使用纯标准库编写的C程序应该可以在任何计算机和操作系统之间移植。

### 1.3.2 第一个POSIX标准

POSIX术语（可移植操作系统接口）表示了一组标准，由电子电气工程协会（IEEE）组织开发，特别是其下属的可移植应用标准委员会（PASC，<http://www.pasc.org/>）。PASC标准的目标是在源代码层面上提高应用的可移植性。

POSIX标准对于我们来说关系最紧密的是第一个POSIX标准，称为POSIX.1（或者全称POSIX 1003.1），以及随后的POSIX.2标准。

**POSIX.1和POSIX.2**

POSIX.1在1988年成为IEEE标准，然后在1990年经过很小的修订，被采纳为ISO标准（ISO/IEC 9945-1:1990）。（原始的POSIX标准没有在线提供，但可以在IEEE的网站<http://www.ieee.org/>上购买）。

POSIX.1定义API提供明确的服务，而且遵循该标准的操作系统必须提供该API。这样的操作系统才可以获得POSIX.1依从的证明。

POSIX.1基于UNIX系统调用和C库函数API，但是并没有要求特定实现一定要与这个接口绑定。这意味着这些接口可以被任何操作系统实现，不必非得是UNIX操作系统。实际上有些厂商已经增加了API到自己私有的操作系统中，获得了依从POSIX.1的证明，同时又大体上保持底层操作系统不变。

原始POSIX.1标准的很多扩展也很重要。1993年通过的IEEE POSIX 1003.1b（POSIX.1b，正式名称是POSIX.4或者POSIX 1003.4），包含了对基础POSIX标准的许多实时扩展。1995年通过的IEEE POSIX 1003.1c（POSIX.1c），定义了POSIX线程。1996年通过了POSIX.1标准的修订版（ISO/IEC 9945-1:1996），核心内容保持不变，但整合了实时与线程扩展。IEEE POSIX 1003.1g（POSIX.1g）定义了网络API，包括socket；1999年通过的IEEE POSIX 1003.1d（POSIX.1d）和2000年通过的POSIX.1j，定义了额外的实时扩展。

另外一个相关的标准，POSIX.2（1992，ISO/IEC 9945-2:1993）标准化了shell和许多UNIX实用工具，包括C编译器的命令行接口。

**FIPS 151-1和FIPS 151-2**

FIPS是联邦信息处理标准的简称，是US政府为采购计算机系统而制定的一组标准。1989年公布了FIPS 151-1。这个标准基于1988年的IEEE POSIX.1标准和ANSI C标准草案。FIPS 151-1和POSIX.1（1988）的主要区别是FIPS标准强制要求某些POSIX.1指定可选的特性。因为US政府是主要的计算机系统采购商，多数计算机厂商都确保自己的UNIX系统遵循FIPS 151-1版本的POSIX.1标准。

FIPS 151-2对应于1990年POSIX.1的ISO版本，其它则保持不变。现在已经过时的FIPS 151-2在2000年2月取消标准。

### 1.3.3 X/Open公司和开放组织

X/Open公司是国际计算机厂商组成的集团，采纳或改编现有标准来产生综合的开放系统标准。它创建了X/Open可移植指南，基于POSIX标准的一系列可移植指南。这个指南的首个重要发布版是1989年的Issue 3（XPG3），随后1992年发布了XPG4，并在1994年重新修订，生成了XPG4的版本2，这个标准同时整合了AT&T System V接口定义Issue 3的重要部分，我们在1.3.7节会再加描述。这个修订版本也被称为Spec 1170，其中1170指的是标准定义的接口数量（函数、头文件、和命令）。

当Novell在1993年初获得了AT&T的UNIX系统业务后（后来又自己丢失了这块业务），把UNIX商标的权利转移给了X/Open（转移的计划发布于1993年，但法律要求延迟到1994初才完成）。XPG4版本2也因此重新包装为Single UNIX Specification（SUS或SUSv1），有时候也叫UNIX 95。包括XPG4版本2、X/Open Curses Issue 4版本2规范、和X/Open网络服务（XNS）Issue 4规范。Single UNIX规范的版本2（SUSv2，<http://www.unix.org/version2/online.html>）发布于1997年，实现并通过验证这个规范就可以称为UNIX 98。（这个标准有时候也被称为XPG5）。

到1996年，X/Open与开放软件基金会合并组成了开放组织。几乎所有与UNIX系统有关联的公司或组织现在都是开放组织的成员，继续开发API标准。

### 1.3.4 SUSv3和POSIX.1-2001

从1999年开始，IEEE、开放组织、和ISO/IEC Joint技术委员会就Austin公共标准修订组织（CSRG，<http://www.opengroup.org/austin/>）进行合作，目标是修订和巩固POSIX标准和Single UNIX规范。（Austin组织由于1998年9月在德克萨斯洲的奥斯丁举行开幕式而得名）。结果在2001年12月批准了POSIX 1003.1-2001，有时候直接称为POSIX.1-2001（随后被采纳为ISO标准ISO/IEC 9945:2002）。

POSIX 1003.1-2001替代了SUSv2、POSIX.1、POSIX.2、和其它早期POSIX标准草案。这个标准也被称为Single UNIX规范版本3，本书后面通常使用SUSv3来引用它。

SUSv3基本规范大概有3700页，分成以下四个部分：

* 基本定义（XBD）：这部分包含定义、术语、概念、和头文件内容规范。一共提供了84个头文件规范。
* 系统接口（XSH）：这部分的开头描述了许多有用的背景信息。中间大部分内容包含许多函数的规范（实现为系统调用或库函数）。这部分总共包含了1123个系统接口）。
* Shell和实用工具（XCU）：这部分规范了shell的操作和许多UNIX命令。总共规定了160个实用工具。
* Rationale（XRAT）：这部分包含与前面几个部分相关联的文本信息和阐述。

此外SUSv3还包含X/Open CURSES Issue 4版本2（XCURSES）规范，规定了curses屏幕处理API相关的372个函数和3个头文件。

SUSv3总共规定了1742个接口。相比较POSIX.1-1990（包含FIPS 151-2）才规定了199个接口，而POSIX.2-1992则规定了130个实用工具。

SUSv3可以在<http://www.unix.org/version3/online.html>上找到。实现并通过SUSv3验证的系统则称为UNIX 03。

原始的SUSv3批准之后，经过了一些小的变化和改进。结果就是Technical Corrigendum Number 1的出现，这些改进最后在2003年被整合到SUSv3修订版，而Technical Corrigendum Number 2的改进则被整合到2004修订版。

**POSIX依从、XSI依从、和XSI扩展**

历史上SUS（和XPG）标准与相应的POSIX标准存在差异，并组织为POSIX的功能超集。除了规定额外的接口，SUS标准还强制要求实现许多POSIX可选的接口和行为。

这种差异在POSIX 1003.1-2001中更为微妙，它同时是IEEE和开放组织技术标准（也是早期POSIX和SUS标准的合并）。这个文档定义了两个级别的依从：

* POSIX依从：定义了依从实现必须提供的接口基准。允许实现提供其它可选接口。
* X/Open系统接口（XSI）依从：要依从于XSI，实现必须符合所有POSIX依从的要求，同时还必须提供许多POSIX可选的接口和行为。实现必须达到这个级别的依从，才能从开放组织获得UNIX 03商标。

XSI依从要求的额外接口和行为合称为XSI扩展。它要求支持的特性包括：线程、mmap()和munmap()、dlopen API、资源限制、伪终端、System V IPC、syslog API、poll()、和登录会计。

在后面章节中，当我们说SUSv3依从时，指的是XSI依从。

**未规定和软规定**

有时候我们会谈到某个接口在SUSv3中“未规定”或“软规定”

对于未规定的接口，意思是虽然可能在背景注解或rationale文本中提到过，但在正式标准中根本没有定义。

对于软规定的接口，则指的是虽然接口包含在标准中，但其重要细节未明确规定（通常是由于委员会成员因现有实现的差异而无法达成一致）。

当使用未规定或软规定的接口时，我们很难保证能够迁移到其它UNIX实现。无论如何，少数情况下这种接口在不同实现间还是比较一致的，这时我们会明确地标注这一点。

**遗留特性**

有时候我们会提到SUSv3标记某个特性是遗留的。这个术语表示这个特性只是为了兼容老的应用而保留，应该避免在新应用中使用。在许多情况下，都有其它API提供等价的功能。

### 1.3.5 SUSv4和POSIX.1-2008

2008年Austin组织完成了POSIX.1和Single UNIX规范的修订。和之前版本的标准一样，它也包含基本规范和XSI扩展。我们把这个修订版称为SUSv4。

SUSv4的变化比SUSv3要少很多。最重要的改变如下：

* SUSv4为一些函数增加了新的规范。在本书中涉及的新规范函数包括：dirfd(), fdopendir(), fexecve(), futimens(), mkdtemp(), psignal(), strsignal(), utimensat()。其它一些文件相关的函数（例如18.11节描述的openat()）是现有函数（如open()）的类似物，区别是它们根据文件描述符来解释相对路径，而不是根据进程的当前工作目录来解释相对路径。
* 有些SUSv3规定为可选的函数在SUSv4中成为强制要求。例如SUSv3中的很多XSI扩展函数现在成为SUSv4的基本标准。这些函数包括dlopen API（42.1节），实时信号API（22.8节），POSIX信号量API（第53章），和POSIX定时器API（23.6节）。
* SUSv3的某些函数被标记为过时。包括asctime(), ctime(), ftw(), gettimeofday(), getitimer(), setitimer(), siginterrupt()。
* 某些SUSv3标记为过时的函数从SUSv4中移除。包括gethostbyname(), gethostbyaddr(), vfork()。
* SUSv3规范的一些细节在SUSv4中进行了修改。例如许多函数被添加到异步信号安全函数列表（表21-1）。

在本书的后面部分，我们会在相关主题被讨论时标注SUSv4的变化。

### 1.3.6 UNIX标准时间线

图1-1总结了前面章节描述的各种标准之间的关系，并按年代顺序排列了所有标准。在这个图中，实线表示标准之间直接继承；而虚线表示某个标准影响了另一个标准，并被整合到另一个标准中，或者推迟为其它标准。

网络标准的情况比较复杂，网络的标准化开始于1980年代末，由POSIX 1003.12委员会标准化socket API、X/Open传输接口（XTI）API（基于System V传输层接口的另一个网络编程API）、以及许多相关的API。这个标准酝酿了很多年，也就是POSIX 1003.12重命名为POSIX 1003.1G期间。最终批准于2000年。

在开发POSIX 1003.1g的同时，X/Open也在开发自己的X/Open网络规范（XNS）。该规范的首个版本XNS Issue 4是首个Single UNIX规范的一部分。后面还有XNS Issue 5，属于SUSv2的一部分。XNS Issue 5和当前的POSIX.1g（6.6）草案本质上是一样的。再后面是XNS Issue 5.2，与XNS Issue 5和POSIX.1g草案不一样的地方是标记XTI API为过时的，并包含了因特网协议版本6（IPv6），后者大约在1990年代中期设计。XNS Issue 5.2组成了SUSv3网络部分的基础，现在已经被废弃。相同的原因，POSIX.1g也很快被废除标准资格。



图1-1：各种UNIX和C标准之间的关系

### 1.3.7 实现标准

除了上面独立或多方组织产生的标准，有时候我们会提到两个实现标准，由最终的BSD发布版（4.4BSD）和AT&T System V发布版4（SVR4）定义。后一个实现标准由AT&T的System V接口定义（SVID）出版而正式化。1989年AT&T出版了SVID Issue 3，定义了UNIX实现要通过System V版本4验证，必须提供的接口。（SVID可以在<http://www.sco.com/developers/devspecs/>上在线查看）。

### 1.3.8 Linux、标准、和Linux标准基础

Linux（内核、glibc、和工具）开发致力于遵循各种UNIX标准，特别是POSIX和Single UNIX规范。但是在本书写作时，还没有哪个Linux发行版获得开放组织的“UNIX”标志。主要的问题是时间和代价。每个厂商的发行版都需要经历依从测试来获得这个认证，而且每个发行版的新版本也需要重新测试。无论如何，Linux能够在UNIX市场上如此成功，得益于Linux对各种标准事实上的接近依从。

对于多数商业UNIX实现来说，相同的公司开发和发行操作系统。而Linux则不同，各个发行版的实现是分开的，而且由多个组织（包括商业和非商业）处理Linux发行版。

Linus Torvalds并没有为某个特定的Linux发行版单独贡献，也没有认可某个Linux发行版。再加上其它个体也在负责Linux的开发，情况就更加复杂了。许多Linux内核和其它自由软件项目的开发者，都受雇于不同的Linux发行版公司，或者工作于对Linux非常感兴趣的公司（如IBM和HP）。这些公司都能够影响Linux的发展方向，但又无法控制Linux的发展。当然Linux内核和GNU项目也有许多贡献者是自愿工作的。

由于存在多个Linux发行版，而且内核实现并不能控制发行版的内容，因此没有“标准”的商业Linux这回事。每个Linux发行商的内核通常都基于内核主版本的某个快照，并应用许多patch而形成。

这些patch一般或多或少都是由于商业需求而提供，目的是提高市场竞争力。有些情况下这些patch后来也被内核采纳。实际上有些新内核特性最初就是由发行公司开发，在成为内核主版本的一部分之前，已经先出现在他们的发行版中。例如Reiserfs日志文件系统版本3先是某些Linux发行版的一部分，然后才被2.4主内核采纳。

上面这些描述的要点是，不同Linux发行版公司提供的系统存在差异（大多数的差异都很小）。从更小的范围来讲，这种实现间的分裂和UNIX早期发生的分裂是一样的。Linux标准基础（LSB）已经在努力，希望能够保证各个Linux发行版之间的可移植性。为了达到这个目标，LSB（<http://www.linux-foundation.org/en/LSB>）开发和推广了一组Linux系统的标准，目的在于确保二进制应用（编译过的程序）可以在任何LSB依从的系统中运行。

## 1.4 小结

UNIX系统最早于1969年在Digital PDP-7微计算机中由贝尔实验室的Ken Thompson实现。UNIX操作系统和它的双关语名字一样，从早期的MULTICS系统中吸收了许多想法。1973年UNIX被移植到PDP-11微计算机中，并用C重新编写，C语言由贝尔实验室的Dennis Ritchie设计和实现。由于法律阻止销售UNIX，AT&T转而向大学发布了完整的系统。这个发布包括源代码，在大学迅速流行起来，因为它提供了便宜的操作系统，并且可以让计算机科学院和学生学习和修改其源代码。

加州大学伯克利分校在UNIX系统的开发中扮演了关键角色。在那里Ken Thompson和一些毕业生扩展了UNIX操作系统。1979年伯克利发布了自己的UNIX系统BSD。这个发布版在学院广泛普及，并成为几个商业实现的基础。

同时AT&T垄断的解体，允许公司开始销售UNIX系统。这就产生了另一个主要的UNIX变种：System V，同样也成为几个商业实现的基础。

两个不同的因素促成了GNU/Linux的开发，其中一个因素是GNU项目，由Richard Stallman成立。到1980年代末，GNU项目已经创建了一个几乎完整的自由UNIX实现。缺少的只是可以工作的内核。1991年，Linus Torvalds受到Andrew Tanenbaum编写的Minix内核的启发，为Intel x86-32体系架构创建了一个可以工作的UNIX内核。Torvalds邀请其它程序员加入，来改进这个内核。许多程序员积极响应，于是Linux被扩展和移植到大量硬件体系架构下。

不同UNIX和C实现在1980年代末存在的可移植性问题，直接促成了标准化进程。1989年C语言标准化（C89），1999年进一步修订标准（C99）。对操作系统接口的首个标准化尝试产生了POSIX.1，并于1988年批准为IEEE标准，1990年批准为ISO标准。在整个1990年代，草拟了许多标准，包括各种版本的Single UNIX规范。2001年POSIX 1003.1-2001和SUSv3结合的标准得到批准。这个标准巩固和扩展了许多早期的POSIX标准和早期的Single UNIX规范。2008年完成了一个不那么广泛应用的标准修订，结合了POSIX 1003.1-2008和SUSv4标准。

和多数商业UNIX实现不同，Linux的实现和发行是分离的。因此没有单一的“官方”Linux发行版。每个Linux发行商都提供当前稳定版内核的某个快照，并增加许多patch。LSB开发和促进了一组Linux系统标准，目的是确保二进制应用在不同Linux发行版之间的可移植性，这样编译后的程序就可以在相同硬件的任何LSB依从系统中运行。

**更多信息**

（略）

# 第2章 基础概念

本章介绍Linux系统编程相关的许多概念。目标是那些主要工作于其它操作系统，或者对Linux和其它UNIX实现只有有限经验的读者。

## 2.1 操作系统的核心：内核

操作系统这个术语通常表示两个不同的意思：

* 表示整个软件包系统，是管理计算机资源的中心软件，包含所有标准软件工具，如命令行解释器、图形用户界面、文件工具、和编辑器。
* 狭义的含义则指管理和分配计算机资源（如CPU、RAM、和设备）的核心软件。

内核这个术语通常则代表第二种意思，本书所说的操作系统也是这种意思。

尽管没有内核也可以在计算机中运行程序，但内核能够极大地简化编写和使用其它程序，并增强程序员的能力和灵活性。内核通过提供软件分层来管理有限的计算机资源。

**内核执行的任务**

内核主要执行以下任务：

* 进程调度：计算机只有一个或少数中央处理单元（CPU）来执行程序指令。和其它UNIX系统一样，Linux是抢先式多任务操作系统，多任务表示多个进程（正在运行的程序）可以同时在内存中，而且每个都可以使用CPU。抢先式表示由内核进程调度器支配哪个进程获得CPU，以及确定进程使用CPU的时间。
* 内存管理：虽然计算机内存容量在近十来年变得非常庞大，但软件的体积也相应地快速增长，因此物理内存（RAM）仍然是一种有限的资源，内核必须以公平和有效的方式使多个进程间共享物理内存。和多数现代操作系统一样，Linux采用了虚拟内存管理机制（6.4节），这个技术有两个主要的优点：
  + 进程与其它进程以及内核隔离，因此一个进程不能读取和修改另一个进程以及内核的内存。
  + 内存中只保留某个进程的部分，因此降低了每个进程的内存需求，允许更多进程同时存在于RAM中。这也提高了CPU利用率，因为增强了这样一种可能性，任何时候至少有一个进程可以让CPU执行。
* 文件系统管理：内核提供文件系统，允许创建、读取、更新、删除文件等等操作。
* 创建和终止进程：内核可以装载新程序到内存中，为其提供运行所需的相关资源（CPU、内存、文件访问等）。每个正在运行的程序就是一个进程。一旦某个进程完成执行，内核确保它使用的资源被释放，并可以提供给接下来的程序使用。
* 设备访问：计算机系统中附加的设备（鼠标、显示器、键盘、磁盘和磁带设备等等）允许计算机与外界进行交流，提供输入和输出功能。内核为程序提供标准化和简化的接口访问设备，同时为多个进程使用设备进行仲裁。
* 网络：内核代表用户进程传输和接收网络信号（包）。这个任务包括将网络包路由至目标系统。
* 提供系统调用应用编程接口（API）：进程可以向内核请求执行不同的任务，使用内核入口也就是系统调用。Linux系统调用API是本书的主要主题。3.1节详细描述了进程执行系统调用时的步骤。

除了上面这些特性，多用户操作系统（如Linux）通常还给用户提供虚拟私有计算机的抽象；每个用户都可以登录到系统中，并与其它用户大体上独立操作。例如每个用户有自己的磁盘存储空间（home目录）。此外用户还可以运行程序，每个程序都能获得共享的CPU，并在自己的虚拟地址空间中操作，这些程序还可以独立的访问设备和通过网络传输信息。内核解决潜在的硬件资源访问冲突，因此用户和进程通常感觉不到冲突的存在。

**内核模式和用户模式**

现代处理器体系架构通常允许CPU至少在两种不同模式下操作：用户模式和内核模式（有时候也称为超级模式）。通过硬件指令就可以在不同模式间切换。相应地虚拟内存也被划分为用户空间和内核空间等区域。当运行在用户模式中时，CPU只能访问标记为用户空间的内存；试图访问内核空间内存会导致硬件异常。当运行在内核模式中时，CPU可以同时访问用户和内核空间内存。

有些操作只有进程处于内核模式时才能执行。例如执行halt指令来停止系统、访问内存管理硬件、发起设备I/O操作等。通过把操作系统放在内核空间中，操作系统实现可以确保用户进程无法访问内核的指令和数据结构，或者阻止用户进程执行有害操作。

**进程VS内核对系统的视角**

在每天的许多编程工作中，我们习惯于按面向进程的方式来思考。但是考虑到本书后面讲解的许多主题，调整我们的视角，从内核的角度来观察会非常有帮助。为了使对比更加明显，我们首先考虑进程视角，然后是内核视角。

一个运行系统通常有许多进程。对于每个进程，很多事情都在异步发生。执行进程并不知道自己什么时候CPU时间用完，其它进程被调度获得CPU，以及自己何时再次被调度，也不知道发生的顺序如何。信号递送和进程间通信事件由内核仲裁，对进程来说可能在任何时间发生。许多事情对进程是透明的。进程不知道自己在RAM中的位置，也不知道自己哪部分内存空间在内存中或是在交换区域（磁盘的保留区域，用来补充计算机的RAM）。类似地，进程也不知道自己访问的文件被存放于磁盘驱动器的位置；进程只是简单地通过名字来引用文件。进程的操作相互独立，不能直接与其它进程通信。进程自己也不能创建新进程，甚至无法终止自己。最后进程也不能直接与计算机的输入输出设备交互。

相比之下，运行系统的内核则知道和控制了所有一切。内核为系统中所有运行进程提供协助。内核决定哪个进程获得CPU访问权，什么时候获得，使用多长时间。内核维护一组进程数据结构，包含所有运行进程的所有信息，并根据进程创建、状态变化、进程终止来更新这些数据结构。内核维护所有底层的文件数据结构，允许程序使用文件名访问文件，并转换为磁盘中的物理位置。内核同时还维护每个进程虚拟内存到物理内存映射，以及到磁盘交换区域映射的数据结构。进程间的所有通信都通过内核提供的机制来完成。根据进程的请求，内核创建新进程或结束现有进程。最后内核（特别是设备驱动）执行所有与输入输出设备的交互，为用户进程传递信息。

本书后面我们讲到“进程可以创建另一个进程”、“进程可以创建管道”、“进程可以向文件写入数据”、“进程可以通过调用exit()终止”，请记住内核仲裁所有这些动作，这些句子只不过是“进程可以请求内核创建另一个进程”的简称。

## 2.2 Shell

Shell是特殊的程序，它读取用户输入的命令，并执行适当的程序来响应这些命令。Shell有时候也被称为命令解释器。

login shell表示用户首次登录时，为运行shell而创建的那个进程。

虽然在某些操作系统中命令解释器是内核的部分，但在UNIX系统中，shell实际上是用户进程。存在许多不同的shell，相同计算机的不同用户可以同时使用不同的shell。比较重要的几个shell如下：

* Bourne shell（sh）：这是被广泛使用的最古老的shell，由Steven Bourne编写。它是UNIX第7版的标准shell。Bourne shell包含许多其它所有shell拥有的特性：I/O重定向、管道、文件名自动生成、变量、环境变量操作、命令替换、后台命令执行、和函数。所有后来的UNIX实现都包含Bourne shell，同时也提供其它某些shell。
* C shell（csh）：这个shell由加州大学伯克利分校的Bill Joy编写。名字的来源是这个shell和C编程语言有许多相似的流控制。C shell提供Bourne shell没有的几个有用的交互特性，包括命令历史、命令行编辑、任务控制、和别名。C shell和Bourne shell不保持向后兼容。尽管BSD的标准交互shell是C shell，shell脚本（马上讲到）通常都是按Bourne shell编写，这样才能在所有UNIX实现中保持可移植。
* Korn shell（ksh）：这个shell由AT&T贝尔实验室的David Korn编写，是Bourne shell的继承者。与Bourne shell保持向后兼容的同时，增加了与C shell类似的交互特性。
* Bourne again shell（bash）：这个shell是GNU项目对Bourne shell的重新实现。提供了类似于C shell和Korn shell的交互特性。bash shell理论上的作者是Brian Fox和Chet Ramey。Bash可能是Linux系统使用最广泛的shell（Linux中Bourne shell是由bash提供的尽可能相似的模拟）。

shell不仅仅为交互用户设计，也可以解释shell脚本，后者是包含shell命令的文本文件。为了实现这个目的，每个shell都有类似于编程语言的机制：变量、循环和条件控制语句、I/O命令、和函数。

每个shell都执行类似的任务，只在语法上存在区别。不管我们讲哪个特定shell的操作，我们通常都只说“shell”，所有shell都按这种方式进行操作。本书的多数例子都需要使用bash，但是除非特别提到，读者可以假设这些例子可以在其它Bourne shell中同样工作。

## 2.3 用户和组

系统的每个用户都有唯一标识，用户可能属于某个或几个组。

**用户**

系统的每个用户都有唯一的逻辑名（用户名）和相应的用户ID（UID数字）。对于每个用户，系统的password文件（/etc/passwd）都有一行对其进行定义，还包含以下额外信息：

* 组ID：数字的组ID，用户加入的第一个组。
* home目录：用户登录后的初始目录。
* 登录shell：用来解释用户命令的shell名称。

这个密码记录可能还包含用户的密码，以加密形式存储。但是由于安全原因，通常密码会存放在单独的shadow密码文件中，只对超级用户可读。

**组**

从管理的角度来讲（特别是控制文件和其它系统资源的访问），把用户组织为组是非常有用的。例如工作于同一个项目的团队成员，需要共享相同的一组文件，就可以把所有成员添加到同一个组。在早期UNIX实现中，用户只能加入一个组。BSD允许用户同时加入多个组，这个想法被其它UNIX实现和POSIX.1-1990标准接受。每个组由系统组文件（/etc/group）一个单独的行定义，主要包括以下信息：

* 组名：组的唯一名称。
* 组ID（GID）：与该组相关联的ID数值。
* 用户列表：逗号分隔的用户登录名列表，这些用户都属于这个组（没有在这里标识的用户也可以在自己的密码文件记录中添加该组）。

**超级用户**

超级用户拥有系统的特别权限。超级用户的用户ID是0，通常登录名是root。在典型的UNIX系统中，超级用户可以绕过系统的所有权限检查。例如超级用户可以访问系统的任何文件，无论文件的权限如何设置；也可以向系统中的任何用户进程发送信号。系统管理员使用超级用户执行许多管理性的任务。

## 2.4 单一目录层次、目录、链接、和文件

内核维护一个单一层次的目录结构，来组织系统中的所有文件。（这和Microsoft Windows明显不同，后者的每个磁盘分区都有自己的目录层次）。层次的最底部是root目录，名为“/”（斜线）。所有文件和目录都是root目录直接或间接的子目录。图2-1显示了这种文件结构的一个例子：



图2-1：Linux单一目录层次的子集

**文件类型**

目录是一种特殊的文件，它的内容是文件名加上相应文件索引的表格。这种文件名+引用的关联就称为链接，而文件可以有多个链接，因此在相同或不同的目录下，可以有文件的多个名字。

目录可以同时包含文件和其它目录的链接。目录之间的链接组成了图2-1所示的目录层次。

每个目录都至少包含两项：“.”（点），链接到目录本身；“..”（点点），链接到父目录，也就是层次中上面那个目录。每个目录（除了root）都有父目录。对于root来说，“..”链接到root目录本身（因此“/..”等同于“/”）。

**符号链接**

和普通链接一样，符号链接也提供名字到文件的映射。但是普通链接是在目录列表中的文件名-指针项，而符号链接则是特殊的文件，它的内容是另一个文件的名字。（换句话说，符号链接有文件名-指针项，指针引用的文件内容是另一个文件的名字）。后一个文件通常称为符号链接的目标，通常也称符号链接“指向”或“引用”目标文件。当在系统调用中指定路径时，多数情况下内核会自动“解引用”（跟随）路径中的每个符号链接，使用实际的文件名替换该符号链接指针。如果符号链接的目标本身也是一个符号链接，那么这个过程可能产生递归。（内核强制解引用的数量限制，以避免环形符号链接）。如果符号链接引用的文件不存在，就称为dangling链接。

通常把普通链接和符号链接分别称为硬链接和软链接。为什么要使用两种类型的链接？我们在后面第18章会做出解释。

**文件名**

在多数Linux文件系统中，文件名最多可以有255个字符长度。文件名可以包含任何字符，除了斜线（/）和null字符（\0）。但是只使用字母和数字，以及“.”（点）、“\_”（下划线）、“-”（连字符）是明智的。这65个字符集[-.\_a-zA-Z0-9]在SUSv3中被称为可移植文件名字符集。

我们应该避免使用不可移植的文件名字符，因为这些字符在shell、正则表达式、或其它上下文中可能有特殊含义。如果一个文件名包含特殊含义的字符，那么这些字符就必须被转义。通常是在前面加上反斜线（\）来表示这些字符不要按特殊含义来解析。在无法使用转义机制的情况下，这个文件名就是不可用的。

我们应该避免以连字符（-）来开始一个文件名，因为这样的文件名可能会被shell错误地解析为命令行参数。

**路径名**

路径名是以可选的“/”开始，包含一系列以“/”分隔的文件名的字符串。除掉最后那个文件名，这串字符就标识了一个目录（或者一个指向目录的符号链接）。路径的最后那个文件名可以是任何文件，也可以是目录。在最后一个“/”之前的所有部分有时候称为路径的目录部分，紧跟最后那个“/”的名字就称为文件，或路径的base部分。

路径名以从左向右的顺序读取；每个文件名都存在于路径名之前那部分所标识的目录。字符“..”可以用在路径名的任何位置，来引用当前位置路径的父目录。

路径名描述了一个文件在单一目录层次架构中的具体位置，可以是绝对或相对路径：

* 绝对路径：开始于“/”，指定了相对于根目录的位置。例如图2-1中的绝对路径：/home/mtk/.bashrc、/usr/include、和/（根目录的路径名）。
* 相对路径：指定相对于进程当前工作目录（下面会介绍）的文件位置，和绝对路径的区别在于不以“/”开始。在图2-1中，相对于目录usr，文件types.h的相对路径就是include/sys/types.h；相对于目录avr，文件.bashrc则可以使用相对路径../mtk/.bashrc来引用。

**当前工作目录**

每个进程都有一个当前工作目录（有时候称为进程的工作目录或当前目录）。这是进程在单一层次目录架构中的“当前位置”，从这个目录开始解析所有的相对路径。

进程继承父进程的当前工作目录。login shell设置自己的当前工作目录为用户密码文件项的home目录。可以使用“cd”命令修改shell的当前工作目录。

**文件所有权和权限**

每个文件都关联到一个用户ID和组ID，定义了该文件的所有权，和文件所属于的组。文件所有权用来确定对于不同用户的访问权限。

要访问一个文件，系统把用户划分为三种类型：文件所有者（user）、文件组ID相匹配的用户（group）、和其它所有用户（other）。每种类型的用户都有三个权限位可以设置（总共9个权限位）：读权限允许读取文件内容；写权限允许修改文件内容；执行权限则允许执行该文件，这个文件要么是程序，要么是某种解释器可以处理的脚本（通常但不一定总是shell）。

这些权限也可以为目录设置，不过含义稍微有些不同：读权限允许列出目录的内容（也就是文件名）；写权限允许修改目录的内容（添加、移除、和修改文件名）；执行权限（有时候称为查找权限）允许访问目录中的文件（还要取决于文件本身的权限设置）。

## 2.5 文件I/O模型

UNIX的I/O模型的一个显著特点就是通用I/O概念。这意味着相同的一组系统调用（open(), read(), write(), close()等等）可以执行所有文件类型的I/O操作，包括设备（内核把应用I/O请求转化为适当的文件系统或设备驱动操作，来执行目标文件或设备的I/O操作）。因此采用这些系统调用的程序可以工作于任何文件类型。

内核本质上只提供一种文件类型，顺序字节流，如果是磁盘文件（磁盘或磁带设备），则可以通过lseek()系统调用进行随机访问。

许多应用和库把换行字符（ASCII码10，有时候也称为linefeed）解释为一行文本的终结符并开始下一行。UNIX系统没有文件结束字符（end-of-file）；读取文件无返回数据时表示到达文件末尾。

**文件描述符**

I/O系统调用通过文件描述符来引用已打开的文件，通常是一个很小的非负整数。文件描述符一般通过调用open()获得，传入路径参数指定要对哪个文件执行I/O操作。

通常进程由shell启动时会继承三个已经打开的文件描述符：描述符0是标准输入，进程把它那里获得输入；描述符1是标准输出，进程向它写入输出数据；描述符2是标准错误，进程向它写入错误信息，并通知异常或错误情况。在交互式shell或程序中，这三个描述符通常都连接到终端。在stdio库中，这三个描述符对应于文件流stdin, stdout, stderr。

**stdio库**

C程序通常采用标准C库中的I/O函数执行文件I/O。这一组函数称为stdio库，包括fopen(), fclose(), scanf(), printf(), fgets(), fputs()等等。stdio函数在I/O系统调用（open(), close(), read(), write()等等）之上。

## 2.6 程序

程序通常有两种存在形式。第一个是源代码，使用编程语言编写（如C），是人类可读的一系列程序语句。程序要被执行，源代码必须转化为第二种形式：二进制机器语言指令，这样计算机才能理解。（这和脚本形成对比，后者是包含许多命令的文本文件，直接由shell或其它命令解释器等程序处理）。程序的这两种含义通常认为是同义的，因为编译和链接最终会将源代码转化为语义相同的二进制机器代码。

**过滤器**

过滤器通常指的是那些从stdin读取输入，执行一些转化后，再将结果写入stdout的程序。例如cat, grep, tr, sort, wc, sed和awk。

**命令行参数**

在C语言中，程序可以访问命令行参数，即程序运行时提供的命令行。要访问命令行参数，程序的main()函数必须如下定义：

int main(int argc, char \*argv[])

argc变量包含命令行参数的总数，单个的参数由argv数组的字符串指针引用。其中第一个字符串argv[0]，标识了程序本身的名字。

## 2.7 进程

最简单地说，进程就是执行中的程序。当程序被执行时，内核装载程序代码到虚拟内存中，为程序变量分配空间，并设置内核数据结构来记录该进程的许多信息（例如进程ID、终止状态、用户ID、和组ID等）。

从内核的视角来看，进程是内核必须为其共享许多计算机资源的实体。由于资源是有限的（如内存），内核一开始只分配一定的资源给进程，然后在进程的生命周期过程中，根据进程的需要和整个系统的负载情况，来调整这些分配。当进程终止时，进程使用的所有资源都会被回收，并提供给其它进程重新使用。其它一些资源（如CPU和网络带宽），还必须在所有进程中公平地共享。

**进程内存布局**

进程逻辑上划分为以下部分，称为段（segment）：

* 文本（Text）：程序的指令。
* 数据（Data：程序使用的静态变量。
* 堆（Heap）：程序可以动态分配额外内存的一个区域。
* 堆栈（Stack）：随着函数调用和返回自动扩展和缩小的一小段内存，为本地变量和函数调用链接信息分配存储空间。

**进程创建和程序执行**

进程可以使用fork()系统调用创建新的进程。调用fork()的进程称为父进程，新创建的进程就是子进程。内核通过复制父进程来创建子进程。子进程获得父进程的数据、堆栈、和堆的拷贝，并且随后可以进行修改，而不影响父进程。（程序的文本，存放于只读内存区域，由父子进程共享）。

调用fork()后，子进程要么执行父进程代码中的另一组函数；或者更常见的是使用execve()系统调用装载和执行一个全新的程序。execve()系统调用销毁现有的文本、数据、堆栈、和堆段，并根据新程序代码的新段进行替换。

有几个C库函数基于execve()系统调用实现，每个都提供稍微不同的接口，但是功能是一样的。所有这些函数都以相同的exec字符串开头，区别在哪里目前并不重要，我们使用exec()来引用所有这些函数。不过要明确一点，Linux中并没有名为exec()的函数。

通常我们使用动词exec来描述execve()和相关库函数执行的操作。

**进程ID和父进程ID**

每个进程都有唯一的整数类型的进程标识符（PID）。每个进程同时还有一个父进程标识符（PPID），标识了创建自己的那个进程。

**进程终止和终止状态**

进程可以按两种方式终止：使用\_exit()系统调用（或者相关的exit()库函数）自己请求终止；或者被信号kill而终止。前一种情况进程会产生一个终止状态，一个很小的非负整数，父进程可以使用wait()系统调用来检查这个值。如果调用\_exit()，进程可以显式地指定自己的终止状态。如果进程被信号杀掉，终止状态根据引起进程终止的信号类型来决定。（有时候我们把传递给\_exit()的参数称为进程的退出状态，以区别于终止状态，后者要么是传递给\_exit()的值，要么是信号kill进程产生的值）。

习惯上终止状态0表示进程成功退出。非0状态表示发生了某种错误。多数shell都可以通过$?变量来获得最后执行程序的终止状态。

**进程用户和组标识符（凭证）**

每个进程都有一组相关的用户ID（UID）和组ID（GID）。包括：

* 实际用户ID和实际组ID：标识进程所属的用户和组。新进程继承父进程的实际用户ID和实际组ID。login shell从系统密码文件相应的域获得实际用户ID和实际组ID。
* 有效用户ID和有效组ID：这两个ID（再加上下面的附加组ID）用来确定进程访问受保护资源时的权限，如文件和进程间通信对象。通常进程的有效ID和相应的实际ID相同。修改有效ID是允许进程获得其它用户和组的权限的一种机制，马上我们就会讲到。
* 附加组ID：这些ID标识进程属于的额外的组。新进程继承父进程的附加组ID。login shell从系统组文件中获取自己的附加组ID。

**特权进程**

在UNIX系统中特权进程的有效用户ID是0（超级用户）。这样的进程可以绕过内核实施的权限限制。相反非特权（或无特权）则是其它用户的进程。这种进程的有效用户ID非0，并且受内核的权限规则控制。

特权进程创建的进程也拥有特权，例如由root用户启动的login shell。另一种使进程拥有特权的方法是通过设置用户ID机制，允许进程使用程序文件拥有者的身份执行该进程。

**能力**

从内核2.2开始，Linux对特权进行了划分。每种特权操作都与特定的能力相关联，只有进程拥有相应的能力，才能执行该特权操作。超级用户进程（有效用户ID等于0）的所有能力都被启用。

赋予进程一组能力子集，可以使其执行某些超级用户才允许的操作，同时又防止其执行其它特权操作。

第30章详细讨论了能力，在本书的后面部分，当提到特定操作只能由特权进程执行时，我们通常会标识出相应的能力。能力的名字以前缀CAP\_开始，例如CAP\_KILL。

**init进程**

系统启动时，内核会创建一个特殊的init进程，它是所有进程的父进程，通常是/sbin/init程序文件。系统中的所有进程都是init或其后代创建的（通过fork()）。init进程的ID总是1，并且以超级用户权限运行。init进程不能被kill（超级用户也不行），只有系统关机时它才会终止。init的主要任务是创建和监控运行系统需要的所有进程（更多细节请参考init(8)手册页）。

**Daemon进程**

daemon是一种特殊用途的进程，daemon的创建和处理与其它进程相同，但是有以下区别：

* 长期运行，daemon进程通常在系统引导时启动，一直运行到系统关机。
* 后台运行，没有控制终端，不能读取输入也无法进行输出。

daemon的典型例子是syslogd，为系统记录日志信息；以及httpd，通过HTTP提供web网页服务。

**环境列表**

每个进程都有一个环境列表，是进程的用户空间内存中维护的一组环境变量。这个列表的每个元素都包含一个名字和相应的值。当通过fork()创建新进程时，继承父进程的环境。因此环境提供了一种父进程向子进程传递信息的机制。当进程使用exec()替换原有程序时，新的程序要么继承老程序的环境，要么使用exec()调用指定的新环境参数。

环境变量在多数shell中都是通过export命令来创建（C shell使用setenv命令），例子如下：

$ export MYVAR='Hello world'

C程序可以使用一个external变量（char \*\*environ）来访问环境，还有许多库函数允许进程获得和修改环境中的值。

环境变量有许多用途。例如shell定义和使用了大量变量，可以被shell执行的脚本和程序访问。包括变量HOME（指定了用户登录目录的路径）、变量PATH（指定了一组目录，shell执行用户输入的命令时会在里面查找相应的程序）。

**资源限制**

每个进程都要消耗资源，例如打开的文件、内存、CPU时间。进程可以使用setrlimit()系统调用设置自己消耗各种资源的上限。每个资源限制都有两个关联的值：软限制，限制了进程可以消耗的资源数量；硬限制，是软限制可以调整的上限。非特权进程可以把软限制设为0到相应的硬限制，但是只能降低硬限制。

当新进程创建时，会继承父进程的资源限制设置。

shell的资源限制可以使用ulimit命令进行调整（C shell使用limit）。这些限制值会被shell执行命令创建的子进程继承。

## 2.8 内存映射

# 第3章 系统编程概念

# 第4章 文件I/O：统一的I/O模型

# 第5章 文件I/O：更多细节

# 第6章 进程

# 第7章 内存分配

# 第8章 用户和组

# 第9章 进程凭证

# 第10章 时间

# 第11章 系统限制和选项

# 第12章 系统和进程信息

# 第13章 文件I/O缓冲

# 第14章 文件系统

# 第15章 文件属性

# 第16章 扩展属性

# 第17章 访问控制列表

# 第18章 目录和链接

# 第19章 监控文件事件

# 第20章 信号：基础概念

# 第21章 信号：信号处理器

# 第22章 信号：高级特性

# 第23章 定时器和睡眠

# 第24章 进程创建

# 第25章 进程结束

# 第26章 监控子进程

# 第27章 程序执行

# 第28章 进程创建和程序执行的更多细节

# 第29章 线程：介绍

# 第30章 线程：同步

# 第31章 线程：线程安全和线程存储

# 第32章 线程：线程取消

# 第33章 线程：更多细节

# 第34章 进程组、会话和任务控制

# 第35章 进程优先级和调度

# 第36章 进程资源

# 第37章 Daemon

# 第38章 编写安全的特权程序

# 第39章 能力

# 第40章 登录会计

# 第41章 共享库基础

# 第42章 共享库高级特性

# 第43章 进程间通信简介

# 第44章 管道和FIFO

# 第45章 System V IPC介绍

# 第46章 System V消息队列

# 第47章 System V信号量

# 第48章 System V共享内存

# 第49章 内存映射

# 第50章 虚拟内存操作

# 第51章 POSIX IPC介绍

# 第52章 POSIX消息队列

# 第53章 POSIX信号量

# 第54章 POSIX共享内存

# 第55章 文件锁

# 第56章 Sockets：介绍

# 第57章 Sockets：UNIX Domain

# 第58章 Sockets：TCP/IP网络基础

# 第59章 Sockets：Internet Domain

# 第60章 Sockets：服务器设计

# 第61章 Sockets：高级主题

# 第62章 终端

# 第63章 可选I/O模型

# 第64章 伪终端

# 附录A：跟踪系统调用

# 附录B：解析命令行参数

# 附录C：转换NULL指针

# 附录D：内核配置

# 附录E：更多信息来源

# 附录F：部分习题解答

# 参考书目

# 索引