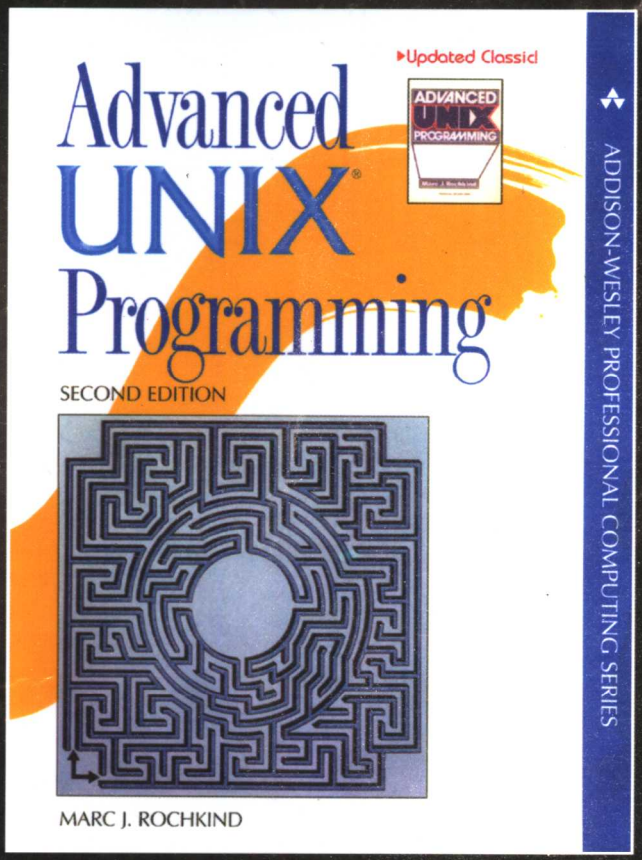
PEARSON Addison Wesley

计 算 机 科 学 丛 书

原书第2版

高级UNIX编程

（美） Marc J. Rochkind 著　王嘉祯 杨素敏 张斌 等译



Advanced UNIX Programming

Second Edition

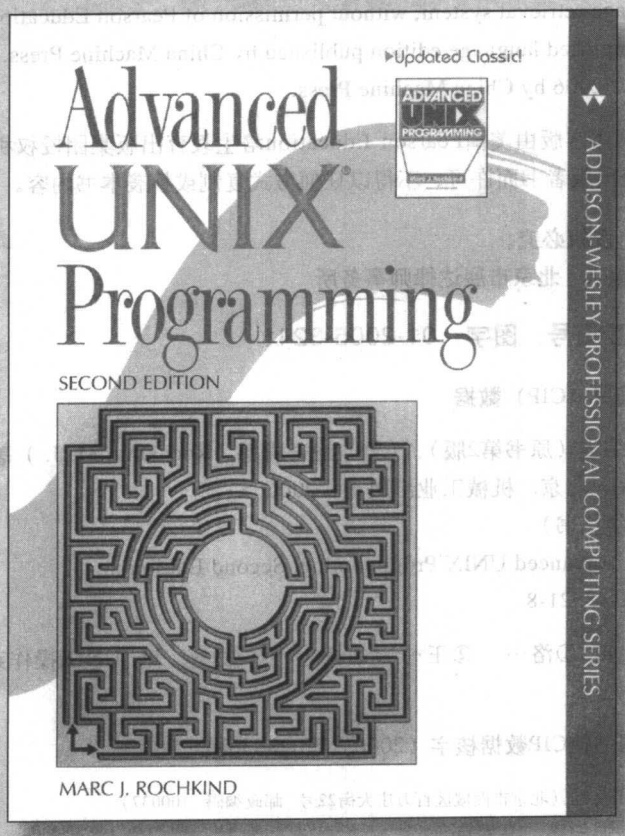
机械工业出版社
China Machine Press

计 算 机 丛 书

原书第2版

高级UNIX编程

(美) Marc J. Rochkind 著　王嘉祯 杨素敏 张斌 等译



Advanced UNIX Programming

Second Edition

机械工业出版社
China Machine Press

本书以当前UNIX规范为基础，详细介绍了UNIX系统函数的用法，并用大量的代码和示例程序进行演示，对实际编程具有指导意义。全书共9章，内容包括：基本概念、基本文件I/O、高级文件I/O、终端I/O、进程与线程、基本进程间通信、高级进程间通信、网络技术与套接字，以及信号与定时器等。涉及POSIX、FreeBSD、Solaris、Linux等几大主流系统实现。每章末都给出一了些练习，一些是简单的程序设计问题，还有一些可以作为学期的UNIX程序设计项目。

本书适合广大UNIX和C程序员、研究人员、高校相关专业师生学习和参考。

本书版权登记号：图字：01-2005-3241

图书在版编目（CIP）数据

高级UNIX编程（原书第2版）/（美）洛奇肯德（Rochkind, M. J.）著；王嘉祯，杨素敏，张斌等译. -北京：机械工业出版社，2006.5

（计算机科学丛书）

书名原文：Advanced UNIX Programming, Second Edition

ISBN 7-111-18521-8

Ⅰ.高…　Ⅱ.①洛…　②王…　③杨…　④张…　Ⅲ.UNIX操作系统-程序设计　Ⅳ.TP316.81

中国版本图书馆CIP数据核字（2006）第028636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：朱起飞　冯春丽

北京牛山世兴印刷厂印刷·新华书店北京发行所发行

2006年5月第1版第1次印刷

787mm × 1092mm 1/16 · 31.25印张

定价：59.00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

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzjsj@hzbook.com
联系电话：（010）68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

译 者 序

UNIX的系统调用是UNIX内核与用户程序之间的接口，是UNIX程序员编写程序时必须要掌握的重要内容。

但是，UNIX规范中大约有1108个系统调用，要想通过规范来学习这些系统调用，不但枯燥无味，而且也是不可能的。本书向读者详细地介绍主要的系统调用，并通过示例代码说明它们的使用方法，再进一步通过练习来实践这些内容。本书是深入学习UNIX、编写应用程序不可多得的优秀教材。

早在1985年，作者就出版了《高级UNIX编程》的第1版，本书是第2版，两版间隔近20年的时间。这期间，UNIX环境发生了很大变化，出现了POSIX、Solaris、Linux、FreeBSD以及Darwin (Mac OS X) 等类UNIX系统。第2版中包括了对以上内容的讨论；除了包含第1版大约70个系统调用之外，又增加了200多个，共有300多个系统调用。这300多个系统调用包括：进程间通信、网络（套接字）、伪终端、高级信号量、实时处理和线程等内容；同第1版一样，本书的第2版还包括了几千行示例代码，其中大多数来源于实际程序（比如shell、全屏菜单系统、Web服务器和实时输出记录器），只是在实际程序的基础上进行了条件简化。这些例子都是用C语言编写的，以帮助读者更好地理解系统调用的含义和具体的应用；此外，在本书的每一章后面都有精心设计的习题，用于巩固所学的内容。这些习题的难度不同，有的习题比较简单，而有的习题则是让读者运用所学的知识进行综合性的练习，有一定难度。

本书的作者Marc J. Rochkind自20世纪70年代开始就在Bell实验室工作，长期致力于UNIX系统的开发研究，积累了许多开发应用系统的经验，本书的第1版出版后就一直为应用程序的编程人员所青睐，现在重新修订的第2版又对UNIX操作系统家族最新的、必需的系统调用进行了系统深入的讲解和示范。

参加本书翻译工作的有：王嘉祯、杨素敏、张斌、冯兵、党辰、彭德云、徐波和胡建理等；王嘉祯、杨素敏等对全书进行了校阅、统稿；中国人民解放军军械工程学院米东教授等为本书的翻译工作提出了许多宝贵的意见和建议，对他付出的辛勤劳动表示深切的谢意。由于书中涉及的知识面比较广泛，再加上译者的水平有限，书中错误和不妥之处在所难免，恳请广大读者批评指正。

王嘉祯

2006年2月

前　言

本书是对1985年版《Advanced UNIX Programming》的更新，覆盖了过去18年来UNIX所发生的一些变化。也许“一些”并不确切！“更新”同样不太确切。的确，除了散落各处的个别句子，本书基本是全新的。第1版大约包含了70个系统调用；本版大约包含了300个。本版所讨论的UNIX标准和实现［POSIX、Solaris、Linux、FreeBSD以及Darwin（Mac OS X）］在1985年左右还未出现。但是我仍然可以在1985年版的前言中找到一些无需修改便可用在这里的句子：

> 本书的主题是UNIX系统调用——即UNIX内核与在其上层运行的用户程序之间的接口。对于那些仅使用命令与系统交互（比如shell、文本编辑器以及其他应用程序）的人来讲，或许不必对系统调用有太多了解，但对于UNIX程序员来说，对系统调用的彻底了解是至关重要的。系统调用是访问诸如文件系统、多任务机制以及进程间通信原语等内核功能的唯一途径。
>
> 系统调用定义了UNIX系统到底是什么。所有的一切（除了子程序和命令行）都是建立在这个基础之上的。尽管这些高层程序的许多新颖之处为UNIX赢得了不少名声，但它们也同样可以在任何现代操作系统上实现。当人们说UNIX系统是个雅致的、简单的、高效的、可靠的和可移植的操作系统时，指的不是其命令（其中一些并不怎么样），而是其内核。

以上所说仍然正确，只是有一点令人遗憾：现在内核的编程接口不再那么雅致、简单了。事实上，由于在过去几十年中UNIX的发展分裂为几个分支，同时因为最初的标准化组织（The Open Group）将几乎所有已有的函数都集合了起来（一共1108个函数），所以导致接口变得笨拙、矛盾、冗余，容易出错和混淆。但它依然高效、可靠并可移植，这就是为什么UNIX和类UNIX系统如此成功的原因。的确，UNIX系统调用接口是迄今我们所拥有的唯一一个具有广泛可移植性的接口，而且这种状况可能在我们有生之年不会改变。

为了理清问题，拥有全部的文档是不够的，就像仅仅拥有黄页并不能找到好的饭店或宾馆一样。我们需要一位向导，能够告诉我们什么是好的、什么是坏的，而不仅仅是告诉我们有哪些东西。这就是本书的目的所在，也是本书与其他UNIX编程书的不同之处。本书不仅要指导读者如何使用系统调用，还要告诉他们不要使用哪些系统调用，因为那些系统调用都不是必需的，它们有的过时了，有的未被正确地实现，有的设计得很糟糕。

下面简单介绍一下本书的大致内容：开篇将介绍单一UNIX规范第3版中定义的1108个函数，但其中去掉了大约590个标准C函数和其他不属于内核接口层的库函数、大约90个POSIX线程函数（保留了其中十多个最为重要的）、大约25个审计登录函数、大约50个跟踪函数、大约15个晦涩废旧的函数以及大约40个用于调度和其他不大有用的函数。本书真正要介绍的只有307个。（见附录D的列表。）不是说这307个全是好的函数——有的也没什么用处，有的甚至还是危险的。但这307个函数都是读者需要了解的。

本书没有包括以下内容：内核实现（除了一些基本的）、设备驱动程序、C程序设计（有些间接的除外）、UNIX命令（shell、vi、emacs等）和系统管理。

全书共有9章：基本概念、基本文件I/O、高级文件I/O、终端I/O、进程和线程、基本的进程间通信、高级进程间通信、网络和套接字以及信号和定时器。先通读第1章，而后就可以自由跳跃浏览了。其中有许多交叉参考，能避免在阅读中迷失。

同第1版一样，这本新书包括了几千行示例代码，其中大多数来源于实际程序（比如shell、全屏菜单系统、Web服务器和实时输出记录器），并进行了简化。这些例子都是用C语言编写的，但在本书的附录B和附录C中给出了其他语言的接口，所以如果你喜欢，就可以采用C++、Java或Jython（Python的变体）来编程。

文字和示例代码仅仅是种资源；实际上还要通过练习来学习UNIX编程。为了提供练习，在每章的末尾都有练习题。这些练习难度不一，有的只需要简单地编写几行代码，有的则是一学期的课程设计。

我选了4种UNIX系统作为详细研究之用，并用来测试例子：Solaris 8、SuSE Linux 8（2.4内核）、FreeBSD 4.6和Darwin（Mac OS X内核）6.8。我将源码保存在FreeBSD系统上，然后用NFS或Samba把代码安装到其他系统上。[⊖]

我在Windows系统上用TextPad编辑代码，使用Telnet、SSH (PuTTY) 或者X Window系统（XFree86和Cygwin）访问4个测试系统。在同一显示屏上打开文本编辑器和4个Telnet/SSH/Xterm窗口十分方便，因为从写代码到在4个系统上测试只需要几分钟时间。另外，我常常使用一个浏览器窗口打开单一UNIX规范，一个浏览器窗口打开Google，一个浏览器窗口运行Microsoft Word写书。除了Word对于像书之类的大型文档（破折号，混合样式，弱交叉引用，古怪的文档组合）有些糟糕之外，所有的工具都很好用。[⊜]我使用Perl和Python做了不同的事情，比如抽取代码样本和维护系统调用的数据库。

所有的示例代码（免费公开代码）、勘误表和更多的内容都在本书的Web站点www.basepath.com/aup上。

我要感谢那些审阅了草稿或者以其他方式提供了技术支持的人：Tom Cargill、Geoff Clare、Andrew Gierth、Andrew Josey、Brian Kernighan、Barry Margolin、Craig Patridge和David Schwartz。另外还要特别感谢那些专心细致地审阅了草稿但要求匿名的人。当然，这些人不需要为您在本书中找到的错误受到谴责——我信任他们。

我还要感谢我的编辑——Mary Franz，是她在一年前提议编写该书的。幸运的是，她正好是在我深入浏览了Linux并再一次为UNIX而兴高采烈的时候找到了我。这使我回想起1972年的时候……

我真心希望您能从本书中得到快乐！如果您发现了错误，或者您将代码移植到了新的系统中，或者您只是想分享您的想法，那么请给我发邮件：aup@basepath.com。

Marc J. Rochkind
Boulder, Colorado
2004年4月

⊖ 这4个系统运行在我多年收集的不同的旧PC和一台我花了200美元在eBay上购买的Mac上。使用SuSE并没有特别的原因，我已经在那台机器上安装了RedHat 9。

⊜ 我本可以采用任何一个系统作为基础系统。Windows用起来很方便，因为我的宽LCD显示器连接在这个系统上，而且我喜欢用TextPad（www.textpad.com）。有关PuTTY的信息请参见站点www.chiark.greenend.org.uk/~sgtatham/putty/。（如果这个链接不管用的话，Google一下“PuTTY”。）

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

目　录

第1章 基本概念

1.1 UNIX和Linux一览

本节将快速浏览UNIX和Linux内核提供的各种功能。这里不会涉及那些通常和UNIX一起提供的用户程序（命令），如`ls`、`vi`和`grep`。对它们的讨论已经超出了本书范围。同时，本书也不会过多涉及有关系统内核的问题（如文件系统是如何实现的）。（从现在起，在书中只要提及UNIX一词，其概念均包含Linux，除非另有说明。）

本节的目的旨在进行一次复习。因为这里假设读者已对某些概念（如*进程*等概念）有了粗略了解，所以在使用这些概念时不会先对其进行定义。如果读者对所提及的概念有些生疏，那就需要在阅读本书前先了解一下UNIX系统（如果还不知道“进程”是什么，那就必须要先热热身！）。目前，有许多UNIX的入门书籍可帮助读者启蒙，其中有两本不错的书：《UNIX开发环境》(The UNIX Programming Environment) [Ker 1984]和《UNIX速成》(UNIX for the Impatient) [Abr 1996]（第2章是一个非常好的入门介绍）。[⊖]

1.1.1 文件

UNIX文件包括以下几种：常规文件、目录、符号链接、特殊文件、命名管道（FIFO）和套接字（socket）文件。这里先介绍前4种文件，后两种将在1.1.7节中介绍。

1.1.1.1 *常规文件*

常规文件（Regular file）包含以线性数组组织的数据字节。任何字节或字节序列都可被读或写。读或写时的开始字节位置由*文件偏移量*（file offset）确定，后者可设定为任意数值（甚至可以超出文件末尾）。另外，常规文件存储在磁盘上。

不能在文件中插入字节（向两端扩展文件）或删除字节（向中间收缩文件），字节将被写入到文件的末尾，每次一个字节，文件长度随之增加。文件可以缩短或增加到任意长度，并可以删除字节或添加零字节。

同一文件可同时被两个或两个以上的进程并发读写。其结果依赖于每个I/O请求发生的次序，且往往不可预见。维护次序的机制可通过文件锁功能和*信号量*（semaphore）来实现，它们是进程可以检测和设置的系统范围内的标志（详见1.1.7节）。

文件没有名称，只有被称为*索引节号*（i-number）的数字。一个索引节号是一批*信息节点*（i-node）的索引，这些信息节点存储在每个包含UNIX文件系统的磁盘区域的前部。每一个信息节点包含关于一个文件的重要信息。有趣的是，这个信息既不包括文件名称，也不包括数据字节。它包括以下内容：文件类型（常规、目录、套接字等）、链接数量（稍后给出解释）、文件所有者ID和组ID、三组访问权限（用户权限、组权限和其他权限）、字节数、最后访问时间、最后修改时间、状态改变时间（信息节点最后被修改的时间），当然还有指向包含文件内容的磁盘块的指针。

⊖ 本书在末尾提供了参考文献。

1.1.1.2 目录和符号链接

由于使用索引节号标识文件不方便，所以引入目录（directory）来使用名称标识文件。在实际中，几乎总是用目录来进行文件访问。

概念上，每个目录包含一个两列表，一列是文件名称，另一列是对应的索引节号。名称/信息节点对被称为链接（link）。当UNIX内核通过名称访问文件时，它会自动查询目录来找寻索引节号，然后获得对应的、包含文件更多信息的信息节点（如谁有权访问该文件）。如果允许访问数据本身，信息节点会给出磁盘上的存储位置。

目录和常规文件非常类似，也占用一个信息节点并拥有数据。因此，在一个目录中，对应于一个特定名称的信息节点也可以是另一个目录中的信息节点。这样就允许用户按照UNIX用户所熟悉的层次结构来安排文件。例如，`memo/july/smith`的路径（path）指示内核获取当前目录的信息节点以确定其数据字节的存储位置，然后从这些数据字节中寻找`memo`，获取对应的索引节号，并获取其信息节点以确定`memo`目录的数据字节位置，再从这些数据字节中找出`july`，获取对应的索引节号，并获取信息节点以确定`july`目录的数据字节位置，最终找到`smith`，并获取对应的信息节点，该信息节点就是和`memo/july/smith`相关联的信息节点。

在对相对路径（relative path）（从当前目录开始的路径）进行追溯时，内核怎样知道应该从何处开始呢？只需要简单地为每个进程保存其当前目录的索引节号即可。当进程改变其当前目录时，它必须提供新目录的路径，该路径对应的索引节号作为新的当前目录的索引节号来保存。

以“/”为开头的绝对路径（absolute path），其起点在根（root）目录。内核为根目录简单地保留了一个索引节号（比方说是2），这在文件系统初次设立时就已建立。通过系统调用可以改变进程的根目录（变为不是2的其他索引节号）。

由于目录的两列结构直接由内核使用（这是内核关心文件内容的少量特例之一），同时因为一个无效目录的存在可能很容易使整个UNIX系统崩溃，所以程序（即使以超级用户权限来执行）不能像常规文件那样写目录。要想对目录进行操作，程序就必须使用一套特定的系统调用。总而言之，合法的写操作仅包括链接的增加和去除。

可以使用同一个索引节号来标识相同或不同目录中的两个或两个以上的链接。这意味着同一文件可能拥有一个以上的名称。但在通过给定路径访问文件时不会产生多义性，因为只会找到一个索引节号。当然，也可以通过其他路径获取索引节号，但结果也是一样的。然而，当在目录中删除某个链接时，不能立刻明确其信息节点以及关联的数据字节是否也可被丢弃。这就是信息节点中包含链接计数的原因。去除指向信息节点的某个链接，只会减少其链接计数；当计数值为零时，内核就删除该文件。

对于为什么不像常规文件一样存在目录的多重链接这一问题，并没有结构上的原因。然而，它会使得扫描整个文件系统命令的编程变得更为复杂，因此，大多数内核将其排除在外。

利用索引节号指向文件的多重链接只在这些链接都属于同一文件系统时才起作用，因为索引节号的唯一性仅限于一个文件系统内部。为避免这种情况带来的后果，还可以采用符号链接（symbolic link），它将要链接的文件路径保存在一个真实文件的数据部分中。这种方式比在某处制作一个第二目录的开销要大，但是它更为常用。用户不需要读和写这些符号链接文件，而仅需使用为符号链接所准备的专门系统调用。

1.1.1.3 特殊文件

特殊文件（special file）通常是某种类型的设备（如CD-ROM驱动或通信链路等）。[⊖]

⊖ 有时，命名管道也被当成是特殊文件，但本书将其另归为一类。

设备特殊文件有两种基本类型：块和字符。*块特殊文件*（block special file）遵循以下特定模型：设备包含固定长度块（如每块4096字节）的数组，并使用内核*缓冲区*（buffer）池作为高速缓冲存储器来加速I/O操作。*字符特殊文件*（character special file）根本不用遵循任何规则，它们进行I/O操作时可以传输很小的块（字符），也可以传输很大的块（磁盘磁道），因此，它们不太规则，不适合使用缓冲区缓存。

同一个物理设备可以同时具有块和字符特殊文件，事实上，磁盘就是这样的物理设备。内核中的文件系统代码通过一个块特殊文件访问常规文件和目录时，可从缓冲区缓存中获益。有时，尤其是高性能的应用程序，需要进行更直接的访问。例如，一台数据库管理器可以完全跳过文件系统，使用一个字符特殊文件访问磁盘（但不是文件系统使用的区域）。大多数UNIX系统拥有这种类型的字符特殊文件，它可以使用*直接存储器访问*（direct memory access，DMA）——一种可以成数量级提高性能的方式，直接在进程的地址空间和磁盘之间传输数据、进行存取。其另一个好处是错误检测的鲁棒性更好，因为缓冲区缓存的间接性会使得错误检测难以实现。

一个特殊文件有一个信息节点，但在磁盘上没有任何指向该信息节点的数据字节。相反，在信息节点的那部分数据字节中包含了一个*设备号*（device number），它是一个列表的索引，该列表是内核用来查找一种叫*设备驱动程序*（device driver）的子程序集合的。

当执行系统调用对特殊文件进行操作时，将调用相应的设备驱动子程序。接着发生的事情完全取决于设备驱动程序的设计者；因为驱动程序是在内核中运行的，而不是作为用户进程运行的，所以它能够访问（甚至修改）任意内核部分、任意用户进程以及计算机本身的任意注册变量或内存。因为向内核中添加新的驱动程序相对简单，所以采用该方式提供一个钩子，不仅可实现与新型的I/O设备进行交互，而且还可做许多事情。一种最流行的办法是，让UNIX做一些它的设计者从未打算让它做的事。想像用认可的方法去做一些很疯狂事情的感觉。

1.1.2 程序、进程和线程

程序（program）是指在磁盘常规文件中存储的*指令*（instruction）和*数据*（data）的集合。在它的信息节点中，文件被标记为可执行的，文件的内容是按照内核建立的规则进行安排的。（这是内核关注文件内容的另一个例子。）

程序员可以选用任何方法创建可执行文件。只要文件内容遵循规则并将文件标记为可执行的，程序就能运行。在实际中，常按如下步骤进行：首先，将使用某种程序设计语言的源程序（如C或C++）输入到常规文件中，通常称为*文本文件*（text file），因为它是按照文本行的方式安排的。其次，创建一个称作*目标文件*（object file）的常规文件，其中包含源程序的机器语言的翻译结果。这项任务由编译器或汇编程序（它们本身也是程序）来完成。如果这个目标文件是完备的（没有缺少子程序），就标识为可执行的，认为其可以运行。如果不完备，就使用*链接器*（linker）（在UNIX的术语中有时叫装载器）来链接这一目标文件和其他那些先前产生的文件，那些先前产生的文件取自一个称为*库*（library）的目标文件的集合。除非链接器找不到它所寻找的东西，它的输出才是完全的、可执行的。[⊖]

为了运行一个程序，首先要请求内核产生一个*新进程*，它是程序运行的环境，由三个部分组成：*指令段*（instruction segment）[⊖]、*用户数据段*（user data segment）和*系统数据段*

⊖ 这不像解释性语言如Java、Perl、Python和shell脚本那样工作。对它们来说，可执行的是解释器，即使将程序编译成某种中间代码，也仅仅是为解释器提供数据，并且UNIX内核决不会见到或关心它们。内核的客户是解释器。

⊖ 在UNIX的专门术语中，将指令段叫做“文本段”，这里是为了避免术语的混淆。

(system data segment)。程序是用来初始化指令和用户数据的。初始化完成后，进程开始脱离运行的程序。虽然现代程序员通常并不修改指令，但会修改数据。另外，进程可能会请求程序没有声明的资源（更多的内存、打开的文件等）。

当进程运行时，内核将记录所有那些对进程数据的相同部分进行读和写的线程。每一个*线程*（thread）是一个单独的指令控制流。（实际上，每一个线程都有自己的堆栈。）在编程时，除非执行特殊的系统调用来创建其他线程，否则只能从一个线程开始。因此，初学者可以认为进程是单线程的。[⊖]

几个并发运行的进程可以由同一个程序初始化。然而，在这些进程之间不存在功能性的关系。通过安排这样的进程来共享指令段，内核也许能节省内存，但参与的进程并不能察觉这样的共享。相反，在同一进程的线程之间具有很强的功能性关系。

进程的*系统数据*（system data）包括诸如当前目录、打开文件的描述符、累计CPU时间等属性。进程不能直接访问或修改它的系统数据，因为它处于进程的地址空间之外。然而，可以通过多种系统调用对这些属性进行访问或修改。

由内核在当前运行进程的基础上产生的新进程称为原进程的*子进程*（child process），原进程称为*父进程*（parent process）。子进程继承了父进程的大多数系统数据属性。例如，如果父进程拥有打开的文件，这些文件对子进程同样处于打开状态。这种类型的继承是UNIX操作的本质，正如本书所要展示的。这不同于创建新线程的线程，同一进程中的线程在大多数方面都是平等的，不存在继承关系。所以线程都平等访问全部数据和资源，而不是它们的副本。

1.1.3 信号

内核可以向进程发送*信号*（signal）。信号可以由内核自身产生，从进程自身发送，从其他进程发送或以用户的名义发送。

例如：段式违例信号是一个由内核发出的信号，是在进程试图访问其地址空间以外的内存时发出的；一个进程发送给自身的信号例子是abort信号，由`abort`函数发出，用来终止一个主存储器信息转储的进程；一个由进程发送给其他进程的信号例子是termination信号，由几个相关进程之一在决定终止所有相关进程时发出；一个由用户发出的信号例子是中断信号，当用户键入Ctrl-c时，发送给由该用户创建的所有进程。

信号的类型约有28种（某些版本的UNIX可能有所出入）。除kill信号和stop信号外，进程对收到的其他信号，能控制响应行为。它既可以接受默认的行为，通常结果是终止进程；也可以忽略信号；还可以在收到信号时，捕获信号并执行某个函数（称为信号处理程序）。信号的类型（如`SIGALRM`）是作为参数传递给处理程序的。然而，处理程序无从得知是谁发送的信号。[⊖]当信号处理程序返回时，进程从断点继续运行。有两种信号没有被内核使用，这可能是应用程序用于其自身的。

1.1.4 进程ID、进程组和会话

每一个进程都有自己的*进程*ID（process-ID），该ID为正整数，在任何时刻都确保是唯一的。除了一个进程之外，其他每个进程都有其父进程。

在进程的系统数据中，同样保留其*父进程*ID（parent-process-ID），即其父进程的进程ID。

⊖ 并不是每一个UNIX版本都支持多线程。多线程是POSIX线程（缩写为“ptread”）选项特征的一部分，它们是在20世纪90年代中期被引入的。关于POSIX更多的内容见1.5节，有关线程的内容见第5章。

⊖ 对于28种基本信号来说是这样的。实时信号选项为某些可用的信息添加了一些信号。详见第9章。

如果子进程的父进程在子进程结束前终止，导致其子进程成为“孤儿”，那么其父进程ID将变为1（或其他固定值）。该值是在启动时创建的初始化进程的进程ID，是所有其他进程的“祖先”。换句话说，初始化进程收留所有的“孤儿”。

在实现子系统时，程序员有时会选用一组相关进程来实现，而不是单个进程。UNIX内核允许将这些相关进程组织成*进程组*（process group），进程组可以进一步组织成*会话*（session）。

会话的成员之一是*会话领导者*（session leader），每个进程组的成员之一是*进程组领导者*（process-group leader）。对于每一个进程，UNIX都会记录其进程组领导者和会话领导者的进程ID，这样，每个进程都能在这种层次结构中找到自己的位置。

内核提供的系统调用是用于向进程组的每个成员发送信号的。通常被用于终止整个组，但任何信号都可以通过这种方式广播。

UNIX外壳程序（shell）一般会为每个登录创建一个会话。它们通常为流水线进程创建一个单独的进程组；在此上下文中，进程组也可称为*任务*（job）。但这只是外壳处理问题的方式；内核允许更灵活的处理方式，使得每个单独的登录都可以建立任意数量的会话，每个会话所拥有的进程组都可以用它喜欢的方式来建立，只是需要保持其层次结构。

其工作方式如下：任何进程都可以脱离原先所在的会话，并成为自身会话（只包含一个进程的进程组）的领导者。然后它可创建子进程并扩充为新进程组。在这个会话中，还可建立其他进程组，并将进程分配到不同的进程组中（这种分配也可改变）。因此，一个单独的用户可能会运行这样一个会话：比如该会话由10个进程构成，而这10个进程又被包含于3个进程组中。

会话和它的进程可以拥有*控制终端*（controlling terminal）——由会话领导者打开的第一个终端设备；这时会话领导者成为*控制进程*（controlling process）。通常，用户进程的控制终端是用户登录所用的终端。当建立新会话时，新会话中的进程不再拥有控制终端，除非已存在一个打开的终端。在一些所谓的*后台程序*（daemon）[⊖]（Web服务器、cron工具等）中，并不存在控制终端，因为这些后台程序一旦启动，就会和启动它们的终端分开运行。

通过组织可以实现会话中仅有一个进程组（即前台任务）可以访问终端，并且也可以在前台和后台之间来回移动进程组，这称为*任务控制*（job control）。在试图访问终端的后台进程组还没有被移至前台之前，先将其挂起。这样可以阻止其非法获取来自前台任务的输入或修改前台进程组的输出。

如果不采用任务控制（一般只有shell使用它），那么会话中的所有进程则都是有效的前台进程，都可以平等地自由访问终端，即使灾难就在前方（随之而来的就是灾难）。

终端设备驱动程序可以发出中断、退出和挂机信号，这些信号从该终端发向以其为控制终端的前台进程组中的每个进程。例如，除非采取预防措施，否则挂断终端（例如，关闭一个`telnet`连接）会终止该组中的所有用户进程。为避免此类情况，可设置进程忽略挂断信号。

另外，无论什么原因，只要会话领导者（控制进程）终止，所有前台进程组中的进程就都会收到一个挂断信号。因此，简单的注销通常会终止用户的所有进程，除非采取特殊措施，或者使它们成为后台进程，或者使它们免受挂断信号的影响。

总之，每个进程都有4个相关的进程ID号：

- 进程ID：能唯一标识这个进程的正整数；
- 父进程ID：其父进程的进程ID；
- 进程组ID：进程组领导者的进程ID。如果和进程ID相同，则该进程即为组的领导者；

⊖ “demon”和“daemon”是同一个单词的两种拼写，但前者的意思是魔鬼，而后者的意思是介于神和人之间的一种超自然的精灵。行为不端的daemon可以称为demon。

• 会话领导者ID：会话领导者的进程ID。

1.1.5 权限

用户ID（user-ID）是和password文件（/etc/passwd）中用户的登录名称（login name）相关联的一个正整数。当用户登录后，login命令会将此ID作为已创建的第一个进程（login shell）的用户ID。其他进程将从shell继承这个用户ID。

用户也会被划分到组（不要和进程组混淆）中，这些组也会有ID，称为组ID（group-ID）。用户的登录组ID被当成其登录外壳（login shell）的组ID。

组是在组文件（/etc/group）中定义的。登录后，用户可以转到他所隶属的其他组中。这样将改变处理请求（通常是shell，通过newgrp命令）的进程的组ID，并被所有派生进程所继承。由于一个用户可能是多个组的成员，所以进程也拥有一个补充组ID列表。对大多数场合来说，在进程的组权限存在争议的情况下，将会检查该列表，使得用户不必经常手工转换组。

这两个用户和组的登录ID被称为实际用户ID（real user-ID）和实际组ID（real group-ID），因为它们代表了实际用户，即登录系统的人员。另两种和进程相关的ID是有效用户ID（effective user-ID）和有效组ID（effective group-ID），它们通常和对应的实际ID相同，但也可能不同，就像稍后所看到的那样。

有效ID经常用于决定权限，真实ID用于审计以及用户到用户的通信。一个表明了用户的权限；另一个表明了用户的身份。

每个文件（常规文件、目录、套接字文件等）的信息节点中都具有一个所有者用户ID（owner user-ID）（简称所有者）和一个所有者组ID（owner group-ID）（简称组）。同时，信息节点中还包含有三组权限位，每组三位（共计9位）。每组包含一个读权限位（read permission bit）、一个写权限位（write permission bit）和一个执行权限位（execute permission bit）。位为1时代表允许权限，为0时代表拒绝权限。三组权限中，一组用于所有者，一组用于组，一组用于其他用户（不在前两类中）。表1-1表明了位的分配（位0是最右位）。

表1-1 权 限 位

位	含 义	位	含 义
8	所有者读	3	组执行
7	所有者写	2	其他用户读
6	所有者执行	1	其他用户写
5	组读	0	其他用户执行
4	组写		

权限位经常用八进制数来表示。例如，八进制数775表示具有所有者及组的读、写和执行权限，但同时只具有其他用户的读和执行权限。ls命令显示其权限组合为rwxrwxr-x；其二进制表示为111111101，直接转化为八进制就是775。（转化为十进制是509，但这种表示通常是没有用的，因为这几个数字和权限位之间没有直观的联系。）

权限系统能够决定指定进程能否对指定文件执行某种预期行为（读、写或执行）。对常规文件来说，所有这三个行为的含义都是明显的。对目录来说，读行为的含义是明显的。目录的“写”权限的含义是指使用系统调用来修改目录（增加或删除链接）的能力。“执行”权限意味着在路径中使用该目录的能力，有时又称为“搜索”权限。对特殊文件来说，读和写权限意味着执行读或写系统调用的能力。其具体含义取决于设备驱动程序的设计者。特殊文件的执行权限是没有任何意义的。