计 算 机 科 学 丛 书

UNIX操作系统设计

The Design of the UNIX Operating System

(美) Maurice J. Bach 著

陈葆珏 王旭 柳纯录 冯雪山 译

机械工业出版社
China Machine Press

Prentice Hall

计算机科学丛书

UNIX 操作系统设计

(美) Maurice J.Bach 著

陈葆珏 王 旭 柳纯录 冯雪山 译

机械工业出版社
China Machine Press

本书以UNIX系统Ⅴ为背景，全面、系统地介绍了UNIX操作系统内核的内部数据结构和算法。本书首先对系统内核结构做了简要介绍，然后分章节描述了文件系统、进程调度和存储管理，并在此基础上讨论了UNIX系统的高级问题，如驱动程序接口、进程间通讯与网络等。

本书可作为大学计算机科学系高年级学生和研究生的教材或参考书，也为从事UNIX系统研究与实用程序开发人员提供了一本极有价值的参考资料。

本书版权登记号：图字：01－2000－0303

图书在版编目（CIP）数据

UNIX操作系统设计/(美)贝奇（Bach.M.J）著；陈葆珏等译.—北京：机械工业出版社，2000.4
(计算机科学丛书)
书名原文：The Design of the UNIX Operating System
ISBN 7－111－07850－0

Ⅰ.U… Ⅱ.①贝…②陈… Ⅲ.操作系统，UNIX-系统设计 Ⅳ.TP316.81

中国版本图书馆CIP数据核字（2000）第03582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：陈 谊
北京第二外国语学院印刷厂印刷·新华书店北京发行所发行
2000年4月第1版·2000年8月第2次印刷
787mm×1092mm 1/16·23印张
印数：6 001-11 000册
定价：35.00元

译 者 序

UNIX操作系统自1974年问世以来，迅速在世界范围内推广。目前，它不仅是小型机、高档微型机、工作站系统的主流操作系统，而且已进入中、大型计算机领地，成为“事实上”的标准操作系统，并被国际标准化组织ISO等考虑和采纳作为分布处理系统的本地操作系统参考模型。值得一提的是，90年代以来随着运行在PC机上的UNIX系统的变体——Linux的出现和迅速普及，使UNIX系统更具生命力。

当前，介绍UNIX系统的书籍很多，然而论述UNIX系统内部结构的专著却屈指可数。本书是其中最引人注目的一本。本书作者Maurice J.Bach多年来在AT&T公司的贝尔实验室工作，对UNIX系统的设计思想有深刻了解，又有讲授UNIX系统的丰富经验。作者在回顾UNIX操作系统的发展演变的基础上，描述了UNIX系统V内核内部的数据结构和算法，并对其做了深入浅出的分析。在每章之后，还给出了大量富有启发性和实际意义的题目。因而，本书不仅可用作大学高年级和研究生操作系统课程的教科书和参考书，也为从事UNIX操作系统的研究人员或UNIX实用程序开发人员提供了极有价值的参考资料。

在本书翻译过程中，我们尽量保持原著的特色，对书中大量的以C伪码形式描述的算法，仍保持C语言的结构和格式。因而，阅读本书的读者应具备一定的C语言基础。另外，对书中的若干明显错误，我们也一一作了修正。有错误或不妥之处，恳请指正。

本书是由北京大学计算机系的几位同志合作翻译的：第1、2、3、4、12章由柳纯录翻译；第5、6、8章由冯雪山翻译；第7、9章由王旭翻译；第10、11、13章由陈葆珏翻译。全书由陈葆珏修改、定稿。特别值得一提的是，本书的翻译从一开始就得到了杨芙清教授的支持和帮助。她还在百忙之中为本书做了校阅，特此表示衷心的感谢。

译者

于北京

前　言

UNIX 系统是由 Ken Thompson 和 Dennis Ritchie 于 1974 年在《ACM 通讯》中的一篇文章中首次提出的[Thompson 74]。从那时起，UNIX 系统得到迅速传播并在计算机工业中得到广泛采用。越来越多的计算机厂家在他们的机器上提供对 UNIX 系统的支持。UNIX 系统在大学里尤其普遍，它通常被用于操作系统的研究及实例分析。

许多专著和文章曾讨论了系统的各个部分，其中有《贝尔系统技术杂志》1978 年和 1984 年的两个专刊[BSTJ 78][BLTJ 84]。还有许多书介绍了 UNIX 系统的用户接口，特别是如何使用电子邮件、如何准备文件及如何使用称为“shell”的命令解释程序等；《The UNIX Programming Environment》[Kernighan 84]（该书已由机械工业出版社引进出版，中译本名为《UNIX 编程环境》）和《Advanced UNIX Programming》[Rochkind 85]等书讨论了程序设计环境。本书则着重描述构成操作系统基础（称为内核）的内部算法和数据结构以及它们与程序员接口之间的关系。因此，本书适用于几种环境。首先，它可用作高年级本科生或一年级研究生的操作系统课程的教材。使用本书的同时若能参考系统源代码则将获益匪浅，但也可以独立地学习本书。其次，系统程序员可将本书作为参考书，从而能更好地理解内核是如何工作的，并可以将 UNIX 系统中采用的算法与其他操作系统的算法加以比较。最后，UNIX 系统上的程序员能够更深入地了解他们的程序是如何与系统相互作用的，从而编出更有效、更高级的程序。

本书的内容及组织形式取自我在 AT&T 贝尔实验室在 1983 年和 1984 年期间讲授的一门课程。尽管这门课集中于阅读系统源代码，但我发现一旦掌握了算法的基本思想，源代码的阅读和理解就会容易得多。在本书里，我已努力使算法的描述尽可能地简单，从而反映出算法所描述的系统的简单性和精巧性。因此，本书并不是用英文逐行地翻译系统；它描述了各种算法的主要流程，更重要的是，它描述了各种算法是如何相互作用的。算法用类似 C 语言的伪码来表示，从而有助于读者理解自然语言的描述；算法的名字对应于内核内部的过程名。书中的各种插图描绘了系统对各种数据结构进行操作时它们之间的关系。在稍后的一些章中，采用了许多小的 C 语言程序来说明一些系统的概念，这些程序用户是容易明白的。为节省篇幅和清晰起见，这些例子一般不检查错误条件，而这一点在写程序时是一定要做的。我已经在系统 V 上运行了这些程序；除了某些演示系统 V 的特殊特点的程序以外，这些程序也应该能在 UNIX 系统的其他版本上运行。

原来为课程所准备的许多习题已放在每章的最后，它们是本书的重要组成部分。有些习题是直接了当的，用于说明正文中引入的概念。有些习题比较困难，用来帮助读者在一个较深的层次上理解系统。最后，还有些习题具有研究性质，设计这些题目是为了提出问题以供研究探讨。难度大的题目都标有 * 号。

本书对 UNIX 系统的描述基于 AT&T 所支持的系统 V，第 2 版。还包括了一些第 3 版的新特点。这是我最熟悉的系统，但我还尽力描述了其他版本对 UNIX 系统的有意义的贡献，特别是 BSD 对系统的修改。本书回避了与特殊的硬件特性有关的问题，力图以通用的

术语描述内核硬件的接口，并忽略特定机器的特殊特点。但是，当与机器有关的问题对理解内核的实现十分重要时，本书则讨论得相对详细一些。至少，对这些问题的探讨会突出操作系统中最依赖于机器的部分。

本书的读者必须具有用高级语言进行程序设计的经验，这是理解本书内容的必备条件，最好还有汇编语言的经验。建议读者具有用 UNIX 系统工作的经验，并了解 C 语言[Kernighan 78]。但是，在编写此书时，我努力使没有这种背景的读者也能理解本书的内容。本书的附录含有系统调用的简单描述，它们足以使读者理解书中的表达方式，但并不作为完整的参考手册。

本书的内容按如下方式组织。第 1 章，系统概貌，简要地描述了用户所看到的系统的特点，并给出了系统结构。第 2 章描述了内核结构的一般概貌，并引入一些基本概念。其余的章节按系统结构所表示的组成部分，描述其中各个成分。这些章可分为三部分：文件系统、进程控制和高级问题。本书先讨论文件系统，因为其概念比进程控制容易一些。这样，第 3 章描述了系统缓冲区高速缓存机制，这是文件系统的基础。第 4 章给出文件系统内部使用的一些算法和数据结构。这些算法使用了第 3 章中解释的算法，并讨论了管理用户文件所需要的内务操作。第 5 章说明提供文件系统用户接口的系统调用；这些系统调用使用了第 4 章的算法来存取用户文件。

第 6 章转向进程控制，其中定义了进程的上下文，讨论了控制进程上下文的内部内核原语。特别地讨论了系统调用接口，中断处理及上下文切换。第 7 章给出了控制进程上下文的系统调用。第 8 章讨论了进程调度问题。第 9 章的内容是存储管理，其中包括对换和请求调页系统。

第 10 章讨论了通用驱动程序接口，特别地讨论了磁盘驱动程序和终端驱动程序。尽管从逻辑上说设备是文件系统的一部分，但是，因为进程控制的问题要在终端驱动程序中出现，所以，对设备的讨论推迟到这一章。这一章也是通向本书其余章节中所给出的更高级的问题的桥梁。第 11 章讨论进程间通信和网络问题，其中包括系统 V 的消息、共享存储区及信号量，还有 BSD 的套接字。第 12 章解释了紧密耦合的多处理机 UNIX 系统。第 13 章研究了松散耦合的分布式系统。

前九章的内容可以在一学期的操作系统课程中完成。其余各章的内容可以在高级讨论班中进行讨论，并同时作各种课题研究。

至此，本人要作几点说明。可以确切地说，本书没有作系统性能方面的讨论，也没有提出任何用于系统安装的配置参数。这些数据会因机器类型、硬件配置、系统版本和实现、以及应用类型等的不同而不同。同时，我有意地尽量避免预测 UNIX 操作系统的未来发展。所讨论的高级问题并不意味着 AT&T 就要提供这些特别的特性，甚至也不意味着那些特殊的领域正在开发研究中。

Maurice J. Bach

目　录

译者序
前言
第1章　系统概貌 …… 1
1.1　历史 …… 1
1.2　系统结构 …… 3
1.3　用户看法 …… 4
1.3.1　文件系统 …… 4
1.3.2　处理环境 …… 8
1.3.3　构件原语 …… 10
1.4　操作系统服务 …… 11
1.5　关于硬件的假设 …… 12
1.5.1　中断与例外 …… 12
1.5.2　处理机执行级 …… 13
1.5.3　存储管理 …… 13
1.6　本章小结 …… 13
第2章　内核导言 …… 15
2.1　UNIX 操作系统的体系结构 …… 15
2.2　系统概念介绍 …… 17
2.2.1　文件子系统概貌 …… 17
2.2.2　进程 …… 19
2.3　内核数据结构 …… 26
2.4　系统管理 …… 27
2.5　本章小结 …… 27
2.6　习题 …… 27
第3章　数据缓冲区高速缓冲 …… 29
3.1　缓冲头部 …… 29
3.2　缓冲池的结构 …… 31
3.3　缓冲区的检索 …… 32
3.4　读磁盘块与写磁盘块 …… 41
3.5　高速缓冲的优点与缺点 …… 43
3.6　本章小结 …… 44
3.7　习题 …… 45
第4章　文件的内部表示 …… 46
4.1　索引节点 …… 46
4.1.1　定义 …… 46
4.1.2　对索引节点的存取 …… 48
4.1.3　释放索引节点 …… 50
4.2　正规文件的结构 …… 51
4.3　目录 …… 55
4.4　路径名到索引节点的转换 …… 56
4.5　超级块 …… 58
4.6　为新文件分配索引节点 …… 59
4.7　磁盘块的分配 …… 64
4.8　其他文件类型 …… 67
4.9　本章小结 …… 67
4.10　习题 …… 68
第5章　文件系统的系统调用 …… 70
5.1　系统调用 open …… 71
5.2　系统调用 read …… 74
5.3　系统调用 write …… 78
5.4　文件和记录的上锁 …… 79
5.5　文件的输入/输出位置的调整——lseek …… 79
5.6　系统调用 close …… 80
5.7　文件的建立 …… 81
5.8　特殊文件的建立 …… 82
5.9　改变目录及根 …… 83
5.10　改变所有者及许可权方式 …… 84
5.11　系统调用 stat 和 fstat …… 85
5.12　管道 …… 85
5.12.1　系统调用 pipe …… 86
5.12.2　有名管道的打开 …… 86
5.12.3　管道的读和写 …… 87
5.12.4　管道的关闭 …… 88
5.12.5　例 …… 89
5.13　系统调用 dup …… 89
5.14　文件系统的安装和拆卸 …… 91
5.14.1　在文件路径名中跨越安装点 …… 94
5.14.2　文件系统的拆卸 …… 97
5.15　系统调用 link …… 98
5.16　系统调用 unlink …… 101
5.16.1　文件系统的一致性 …… 102
5.16.2　竞争条件 …… 103
5.17　文件系统的抽象 …… 106

5.18 文件系统维护 …… 107
5.19 本章小结 …… 108
5.20 习题 …… 108
第6章 进程结构 …… 113
6.1 进程的状态和状态的转换 …… 113
6.2 系统存储方案 …… 116
6.2.1 区 …… 117
6.2.2 页和页表 …… 118
6.2.3 内核的安排 …… 120
6.2.4 u区 …… 121
6.3 进程的上下文 …… 122
6.4 进程上下文的保存 …… 124
6.4.1 中断和例外 …… 124
6.4.2 系统调用的接口 …… 126
6.4.3 上下文切换 …… 129
6.4.4 为废弃返回 (abortive return) 而保存上下文 …… 131
6.4.5 在系统和用户地址空间之间拷贝数据 …… 131
6.5 进程地址空间的管理 …… 132
6.5.1 区的上锁和解锁 …… 132
6.5.2 区的分配 …… 132
6.5.3 区附接到进程 …… 133
6.5.4 区大小的改变 …… 134
6.5.5 区的装入 …… 136
6.5.6 区的释放 …… 137
6.5.7 区与进程的断接 …… 138
6.5.8 区的复制 …… 139
6.6 睡眠 …… 140
6.6.1 睡眠事件及其地址 …… 140
6.6.2 算法 sleep 和 wakeup …… 141
6.7 本章小结 …… 144
6.8 习题 …… 145
第7章 进程控制 …… 147
7.1 进程的创建 …… 147
7.2 软中断信号 …… 153
7.2.1 软中断信号的处理 …… 156
7.2.2 进程组 …… 161
7.2.3 从进程发送软中断信号 …… 162
7.3 进程的终止 …… 163
7.4 等待进程的终止 …… 165
7.5 对其他程序的引用 …… 167
7.6 进程的用户标识号 …… 175
7.7 改变进程的大小 …… 177
7.8 shell 程序 …… 179
7.9 系统自举和进程 init …… 181
7.10 本章小结 …… 184
7.11 习题 …… 185
第8章 进程调度和时间 …… 192
8.1 进程调度 …… 192
8.1.1 算法 …… 192
8.1.2 调度参数 …… 192
8.1.3 进程调度的例子 …… 196
8.1.4 进程优先权的控制 …… 198
8.1.5 公平共享调度 …… 198
8.1.6 实时处理 …… 200
8.2 有关时间的系统调用 …… 200
8.3 时钟 …… 203
8.3.1 重新启动时钟 …… 203
8.3.2 系统的内部定时 …… 203
8.3.3 直方图分析 …… 205
8.3.4 记帐和统计 …… 208
8.3.5 计时 …… 208
8.4 本章小结 …… 208
8.5 习题 …… 209
第9章 存储管理策略 …… 211
9.1 对换 …… 211
9.1.1 对换空间的分配 …… 211
9.1.2 进程的换出 …… 214
9.1.3 进程的换入 …… 217
9.2 请求调页 …… 221
9.2.1 请求调页的数据结构 …… 222
9.2.2 偷页进程 …… 227
9.2.3 页面错 …… 230
9.2.4 在简单硬件支持下的请求调页系统 …… 235
9.3 对换和请求调页的混合系统 …… 237
9.4 本章小结 …… 237
9.5 习题 …… 238
第10章 输入/输出子系统 …… 241
10.1 驱动程序接口 …… 241
10.1.1 系统配置 …… 242
10.1.2 系统调用与驱动程序接口 …… 243
10.1.3 中断处理程序 …… 249
10.2 磁盘驱动程序 …… 250

10.3 终端驱动程序 …… 253
10.3.1 字符表 clist …… 255
10.3.2 标准方式下的终端驱动程序 … 256
10.3.3 原始方式下的终端驱动程序 … 262
10.3.4 终端探询 …… 262
10.3.5 建立控制终端 …… 264
10.3.6 间接终端驱动程序 …… 265
10.3.7 注册到系统 …… 265
10.4 流 …… 266
10.4.1 流的详细的示例 …… 269
10.4.2 对流的分析 …… 270
10.5 本章小结 …… 271
10.6 习题 …… 272
第 11 章 进程间通信 …… 274
11.1 进程跟踪 …… 274
11.2 系统Ⅴ IPC …… 277
11.2.1 消息 …… 278
11.2.2 共享存储区 …… 284
11.2.3 信号量 …… 288
11.2.4 总的评价 …… 295
11.3 网络通信 …… 295
11.4 套接字 …… 296
11.5 本章小结 …… 301
11.6 习题 …… 301
第 12 章 多处理机系统 …… 303
12.1 多处理机系统的问题 …… 303
12.2 主从处理机解决方法 …… 304
12.3 信号量解决方法 …… 306
12.3.1 信号量定义 …… 307
12.3.2 信号量实现 …… 307
12.3.3 几个算法 …… 314
12.4 Tunis 系统 …… 317
12.5 性能局限性 …… 318
12.6 习题 …… 318
第 13 章 分布式 UNIX 系统 …… 320
13.1 卫星处理机系统 …… 321
13.2 纽卡斯尔连接 …… 328
13.3 透明型分布式文件系统 …… 330
13.4 无存根进程的透明分布式模型 …… 333
13.5 本章小结 …… 334
13.6 习题 …… 334
附录 A 系统调用 …… 337
参考文献 …… 353
索引 …… 356

第1章 系统概貌

UNIX 系统自从 1969 年问世以来已经变得相当流行，它运行在从微处理机到大型机的具有不同处理能力的机器上，并在这些机器上提供公共的执行环境。UNIX 系统可分成两部分，第一部分由一些程序和服务组成，其中包括 shell 程序、邮件程序、正文处理程序包以及源代码控制系统等，正是这些程序和服务使得 UNIX 系统环境如此受欢迎，它们是用户立即可见的部分。第二部分由支持这些程序和服务的操作系统组成。本书给出了该操作系统的详细描述，它着重描述由美国电话电报公司（AT&T）生产的 UNIX 系统 V，但也考虑了其他版本所提供的颇有意义的特征。它考察了在该操作系统中使用的主要数据结构和算法，而这些数据结构和算法最终向用户提供了标准用户界面。

本章是 UNIX 系统的引言，它回顾了 UNIX 系统的历史并勾画出了整个系统结构的轮廓。下一章将对该操作系统做更详细的介绍。

1.1 历史

1965 年，贝尔电话实验室和通用电气公司及麻省理工学院的 MAC 课题组一起联合开发一个被称为Multics[Organick 72]的新操作系统。Multics 系统的目标是要向大的用户团体提供对计算机的同时访问，支持强大的计算能力与数据存储，以及允许用户在需要的时候容易地共享他们的数据。贝尔实验室中后来参加 UNIX 系统早期开发的许多人当时都参加了 Multics 工作。虽然 Multics 系统的原始版本于 1969 年在 GE645 计算机上运行了，但它既没能提供预定的综合计算服务，而且，连它自己也不清楚究竟什么时刻算达到开发目标了。结果，贝尔实验室退出了这一项目。

在他们结束了 Multics 工程上的工作的时候，贝尔实验室计算科学研究中心的成员们处于缺乏“方便的交互式计算服务”的境况之中[Ritchie 84a]。为了改善他们的程序设计环境，Ken Thompson、Dennis Ritchie 及其他人勾画出了一个纸面上的文件系统设计——它后来就演化为 UNIX 文件系统的早期版本。Thompson 编写了若干程序，模拟所建议的文件系统行为，以及模拟在请求调页环境下程序的行为。他甚至为 GE645 计算机的简单内核进行了编码。与此同时，他用 Fortran 语言为 GECOS 系统（Honeywell635）编写了名为“宇宙旅行”的游戏程序。但这个程序是不能令人满意的，因为它很难控制“宇宙飞船”，并且该程序运行开销太大。Thompson 后来发现了一个几乎无人问津的 PDP-7 计算机能提供很好的图形显示和廉价的执行开销。为 PDP-7 开发“宇宙旅行”程序使 Thompson 学到了关于该机器的细节，但是它的程序开发环境要求先在 GECOS 机上进行程序的交叉汇编，而后把纸带带到 PDP-7 上输入。为了创建一个较好的开发环境，Thompson 和 Ritchie 在 PDP-7 上实现了他们的系统设计，其中包括 UNIX 文件系统、进程子系统的早期版本及少量实用程序。终于，新系统再也不需要把 GECOS 系统作为开发环境，而能够自己支持自己了。这个新的系统被命名为 UNIX。UNIX 是针对 Multics 的双关语，它是计算科学研究中心的另一名成员 Brian Kernighan 想出来的。

虽然 UNIX 系统的早期版本是大有前途的，但直到它用于实际项目之前它并没能发挥出它的潜力。因此，为给贝尔实验室的专利部门提供一个正文处理系统，1971 年 UNIX 系统被移植到 PDP-11 上。该系统的特征是它的规模小：内存中 16K 字节用于系统，8K 字节用于用户程序；磁盘 512K 字节，每个文件限定长度为 64K 字节。在它初次成功之后，Thompson 开始动手为这个系统实现 Fortran 编译程序。但是在BCPL[Richards 69]的影响下开发出来的却是 B 语言。B 语言是解释性语言，在性能上有所退步——这是这类语言的共同特征。因此，Ritchie 把 B 发展成他称之为 C 的语言，C 语言允许产生机器代码、说明数据类型及定义数据结构。1973 年，用 C 语言重写了 UNIX 操作系统。这一步在当时并不太引人注目，但对其外部用户接受它却具有极大的影响。这之后，贝尔实验室的装机数目增加到 25 个，并且形成了一个 UNIX 系统小组，以提供内部支持。

由于美国电话电报公司 1965 年与联邦政府签署了反垄断法，所以这时它不能销售计算机产品。但是，美国电话电报公司把 UNIX 系统提供给了请求把 UNIX 用于教育目的的大学。该公司信守了反垄断法的条款，它既没有为 UNIX 系统做广告，也没有销售和支持 UNIX 系统。然而，UNIX 系统的声望却在稳步增长。1974 年，Thompson 与 Ritchie 在《ACM 通讯》上发表了一篇描述 UNIX 系统的文章[Thompson 74]，进一步促进了它的可接受性。到 1977 年，UNIX 系统的装机数目已经增长到大约 500 个，其中有 125 个在大学。UNIX 系统开始在业务电话公司流行起来，为程序开发、网络事务操作服务及实时服务（通过 MERT[Lycklama 78a]）提供了良好的环境。这时，UNIX 系统的许可证被提供给商业机构，同时也向大学提供。1977 年，交互系统公司（Interactive Systems Corporation）成了UNIX系统的第一个增值转卖商（VAR）㊀，他们增强了它，使之在办公室自动化环境中使用。同样，1977 年也是标志 UNIX 系统首次被“移植”到非 PDP 机（即稍加改变或完全不变而在其他机种上运行）—— Interdata 8/32 上的一年。

随着微处理机的日益普及，其他公司也把 UNIX 系统移植到新的机器上，但是它的简单清晰的特点吸引着很多开发者以他们自己的方式增强 UNIX 系统，结果在基本系统上产生若干变体。从 1977 年到 1982 年这一时期，贝尔实验室把若干 AT&T 变体综合成一个单个系统，这就是大家都知道的商用 UNIX 系统Ⅲ。随后，贝尔实验室又把若干特征加到系统Ⅲ上，称为新产品 UNIX 系统Ⅴ㊁，1983 年 1 月，美国电话电报公司发布它对系统Ⅴ的正式支持。然而，加利福尼亚大学伯克利分校的人们已经开发了一个 UNIX 系统的变体，它的最新版本称为 4.3BSD（Berkeley Software Distribution），是配在 VAX 机上的，它提供了一些新的有意义的特征。本书将着重描述 UNIX 系统Ⅴ，但也偶尔谈及 BSD 系统中所提供的那些特征。

到 1984 年初，世界上大约安装了 100 000 个 UNIX 系统，它们运行在从微处理机到大型机的具有宽广范围的计算能力的机器上，运行在出自不同的制造厂家的生产线的机器上。没有任何其他操作系统能与之匹敌。UNIX 系统的普及与成功可归结为如下一些原因：

- 该系统以高级语言书写，使之易读、易懂、易修改、易移植到其他机器上。据

㊀ 增值转卖商把具体应用加到计算机系统上以满足特定的市场需要。他们销售的是应用而不是销售这些应用赖以运行的操作系统。

㊁ 对系统Ⅳ发生了什么？事实上，系统Ⅳ是 UNIX 系统的一个内部版本，它被融化到系统Ⅴ中去了。

Ritchie估计，用C语言书写的第一个系统与用汇编语言书写的系统相比，要大而且慢20—40%。但是，采用高级语言的优点要远远超过它的缺点（见[Ritchie 786]第1965页）。

- 它有一个简单的用户界面但具有提供用户所希望的服务的能力。
- 它提供能够由较简单的程序构造出复杂程序的原语。
- 它使用了在维护上是容易的、在实现上是高效的层次式文件系统。
- 文件采用字节流这样的一致格式，使应用程序易于书写。
- 它为外围设备提供了简单一致的接口。
- 它是一个多用户、多进程系统，每个用户都能同时执行几个进程。
- 它向用户隐蔽了机器的体系结构，使用户易于书写在不同硬件实现上运行的程序。

简单性与一致性突出了UNIX系统的宗旨，上面列出的大部分原因都讲的是简单性与一致性。

虽然操作系统和很多命令程序是用C语言书写的，但是UNIX系统支持其他语言，包括Fortran、Basic、Pascal、Ada、Cobol、Lisp及Prolog等。UNIX系统能支持具有编译程序或解释程序的任何语言；UNIX系统还能支持一个系统接口，该接口把用户对操作系统服务的请求映射到UNIX系统使用的一组标准请求上。

1.2 系统结构

图1-1绘出了UNIX系统的高层次的体系结构。该图中心的硬件部分向操作系统提供将在1.5节中描述的基本服务。操作系统直接[⊖]与硬件交互，向程序提供公共服务，并使它们同硬件特性隔离。当我们把整个系统看成层的集合时，操作系统通常称为系统内核，或简称内核（kernel），此时强调的是它同用户程序的隔离。因为程序是不依赖于其下面的硬件的，所以，如果程序对硬件没有做什么假定的话，就容易把它们在不同硬件上运行的UNIX系统之间搬动。比如，那些假定了机器字长的程序比起没假定机器字长的程序来就较难于搬到其他机器上。

外层的程序，诸如shell及编辑程序（ed与vi），是通过引用一组明确定义的系统调用而与内核交互的。这些系统调用通知内核为调用程序做各种操作，并在内核与调用程序之间交换数据。图中出现的一些程序属于标准的系统配置，就是大家知道的命令，但是，由名为a.out的程序所指示的用户私用程序也可以存在于这一层。此处的a.out是被C编译程序产生的可执行文件的标准名字。其他应用程序能在较低层的程序之上构建，因此它们存在于本图的最外层。比如，标准的C编译程序cc就处在本图的最外层：它调用C预处理程序、两遍编译程序、汇编程序及装入程序（称为连接-编辑程序），这些都是彼此分开的低层程序。虽然该图对应用程序只描绘了两个级别的层次，但用户能够对层次进行扩充，直到级别的数目适合于自己的需要。确实，为UNIX系统所偏爱的程序设计风格鼓励把现存程序组合起来去完成一个任务。

⊖ 在UNIX系统的某些实现中，该操作系统同一个核（native operating system）接口，再由核与其下面的硬件接口，并向UNIX系统提供必要的服务。这样的配置可以使其他操作系统及其应用程序能够同UNIX系统平行地运行在装置上。这样的一种配置的经典例子是MERT系统[Lycklama 78a]。更近期的配置包括面向IBM系统/370计算机[Felton 84]及面向UNIVAC1100系列计算机[BodenStab 84]的实现。

一大批提供了对系统的高级看法的应用子系统及应用程序，诸如 shell、编辑程序、SCCS（source code control system）及文档准备程序包等，都逐渐变成了“UNIX 系统”这一名称的同义语。然而，它们最终都使用由内核提供的低层服务，并通过系统调用的集合来利用这些服务。系统 V 中大约有 64 个系统调用，其中将近 32 个是常用的。它们有简单的可选项，这些可选项使系统调用容易使用，但是却向用户提供了很多能力。系统调用的集合及实现系统调用的内部算法形成了内核的主体，因而本书所要介绍的对 UNIX 操作系统的研究，就化为对系统调用及其相互作用的详细研究和分析。简言之，内核提供了 UNIX 系统全部应用程序所依赖的服务，它也定义了这些服务。本书将频繁使用“UNIX 系统”、“内核”或“系统”等术语，但其含义是指 UNIX 操作系统的内核，并且在上下文中是清楚的。

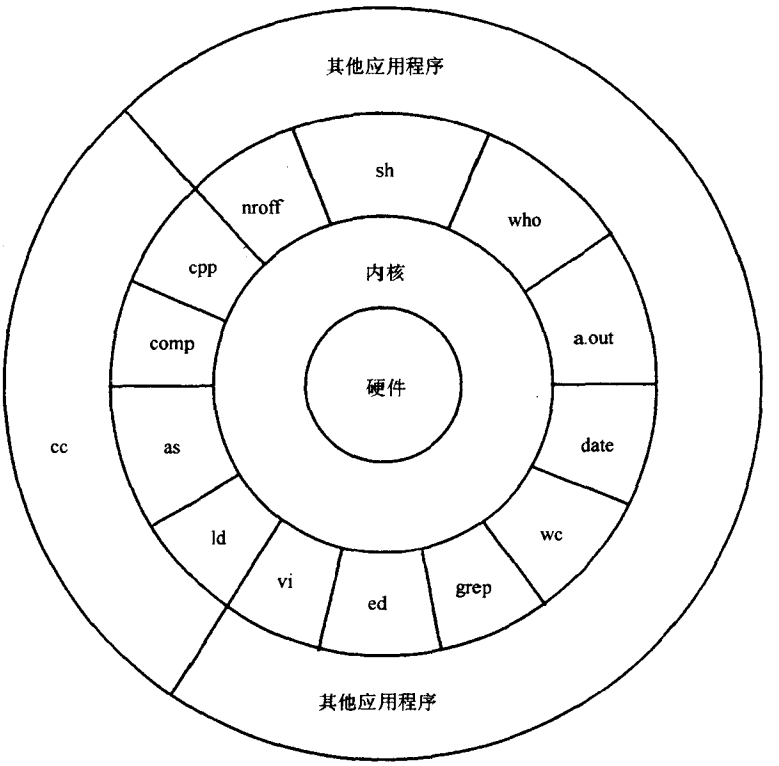

图 1-1 UNIX 系统的体系结构

1.3 用户看法

本节简要地考察 UNIX 系统的高层特征，诸如文件系统、处理环境及构件原语（比如，管道），后面几章将详细地探讨内核对这些特征的支持。

1.3.1 文件系统

UNIX 文件系统有如下特点：

- 层次结构。
- 对文件数据的一致的处理。

- 建立与删除文件的能力。
- 文件的动态增长。
- 文件数据的保护。
- 把外围设备（如终端及磁带装置）作为文件对待。

文件系统被组织成树状，树有一个称为根（root）的节点（记作“/”）。文件系统结构中的每个非树叶节点都是文件的一个目录（directory），树的叶节点上的文件既可以是目录，也可以是正规文件（regular files），还可以是特殊设备文件（special device files）。文件名由路径名（path name）给出，路径名描述了怎样在一个文件系统树中确定一个文件的位置。路径名是一个分量名序列，各分量名之间用斜杠符隔开。分量是一个字符序列，它指明一个被唯一地包含在前级（目录）分量中的文件名。一个完整的路径名由一个斜杠字符开始，并且指明一个文件，这个文件可以从文件系统的根开始，沿着该路径名的后继分量名所在的那个分支游历文件树而找到。因此，路径名“/etc/passwd”，“/bin/who”及“/usr/src/cmd/who.c”都是图 1-2 的树中的文件，但“/bin/passwd”及“/usr/src/date.c”则不是。一个路径名不一定非从根开始不可，可以省略掉路径名中的初始斜杠，由相对于正在执行的进程的当前目录来指明。因此，若从目录“/dev”开始，路径名“tty01”标明的是整个路径名为“/dev/tty01”的文件。

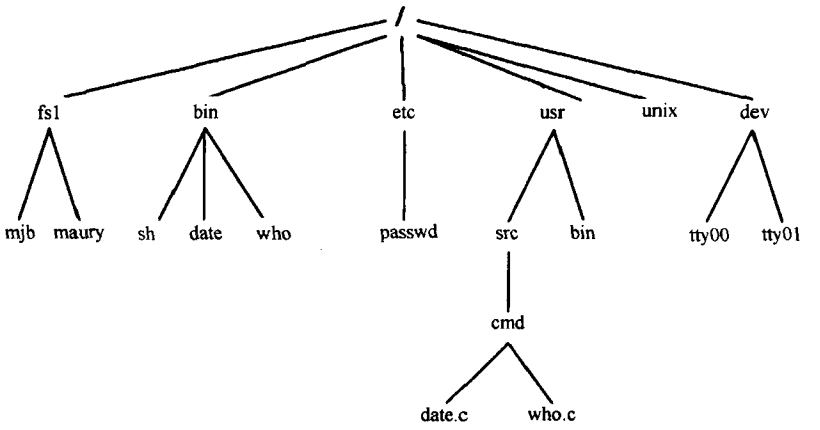

图 1-2　文件系统树示例

在 UNIX 系统中，程序不了解核心按怎样的内部格式存储文件，而把数据作为无格式的字节流看待。程序可以按它们自己的意愿去解释字节流，但这种解释与操作系统如何存储数据无关。因此，对文件中数据进行存取的语法是由系统定义的，并且对所有的程序都是同样的。但是，数据的语义是由程序加上去的。比如，文本格式化程序 troff 希望在文本的每一行的尾部找到“换行”符，而系统记帐程序 acctcom 则希望找到定长记录。两个程序都使用相同的系统服务，以存取文件中的作为字节流存在的数据，而在内部，它们通过分析把字节流解释成适当的格式。如果哪一个程序发现格式是错误的，则它负责采取适当的动作。

从这方面说，目录也像正规文件。系统把目录中的数据作为字节流看待。但是，由于该数据中包含着许多以预定格式记录的目录中的文件名，所以，操作系统以及诸如 ls（列出文件名和属性）这样的程序就能够在目录中发现文件。

对一个文件的存取许可权由与该文件相联系的 access permissions 所控制。存取许可权能够分别对文件所有者、同组用户及其他人这三类用户独立地建立存取许可权，以控制读、写及执行的许可权。如果目录存取许可权允许的话，则用户可以创建文件。新创建的文件是文件系统目录结构的树叶节点。

对于用户来说，UNIX 系统把设备看成文件。以特殊设备文件标明的设备，占据着文件系统目录结构中的节点位置。程序存取正规文件时使用什么语法，它们在存取设备时也使用什么语法。读、写设备的语义在很大程度上与读、写正规文件时相同。设备保护方式与正规文件的保护方式相同：都是通过适当建立它们的（文件）存取许可权实现的。由于设备名看起来像正规文件名，并且对于设备和正规文件能执行相同的操作，所以大多数程序在其内部不必知道它们所操纵的文件的类型。

例如，考察图 1-3 中为一个现存的文件创建新拷贝的程序。假设该程序的可执行版本的名字是 copy。终端上的用户通过敲入：

```
copy oldfile newfile
```

调用该程序。此处，oldfile 是一个现存文件名，而 newfile 是一个新文件名。系统引用 main 时需要提供 argc 作为表 argv 中的参数个数，并且对数组 argv 的每个成员赋初值，以指向由用户提供的参数。在上述例子中，argc 为 3，argv[0]指向字符串 copy（按惯例，第 0 个参数是程序名），argv[1]指向字符串 oldfile，argv[2]指向字符串 newfile。然后该程序检查它被调用时调用者提供的参数个数是否正确。如果正确，则调用系统调用 open，试图打开文件 oldfile，并对该文件做“只读”操作。如果该系统调用成功，则进一步调用系统调用 creat，以便创建 newfile。新创建的文件的存取权限将是 0666（八进制），即允许所有的用户读写该文件。所有的系统调用在失败时都返回 -1。如果 open 或 creat 调用失败，则该程序打印出一则消息，并以返回状态值“1”调用系统调用 exit，结束程序的执行并指出有些地方出错了。

系统调用 open 与 creat 返回一个称为文件描述符（file descriptor）的整数，程序随后就使用这一文件描述符访问该文件。接着，程序调用子程序 copy，copy 进入一个循环：调用系统调用 read，从现存文件中读满一缓冲区的字符，并调用系统调用 write，把这些数据写到新文件上。系统调用 read 返回所读的字节数，当它到达文件尾时返回 0。该程序在遇到文件尾时结束循环，或者在系统调用 read 出了什么错误时（它不检查写错误）结束循环。然后，它从 copy 返回并以返回状态值“0”调用系统调用 exit，以指示该程序成功地结束了。

该程序可以拷贝作为参数提供给它的任何文件——只要它对现存文件具有打开许可权以及对新文件具有建立许可权。此处所说的文件可以是一个可打印字符的文件，比如程序的源代码；或者，它包含不可打印的字符，甚至该程序本身。因此，两个调用

```
copy copy.c newcopy.c
copy copy newcopy
```

都工作。老文件也可以是一个目录，比如

```
copy.dircontents
```

```
#include <fcntl.h>
char buffer[2048];
int version = 1;        /* 第2章对此做了解释 */

main(argc,argv)
    int argc;
    char *argv[];
{
    int fdold, fdnew;

    if(argc != 3)
    {
        printf("need 2 arguments for copy program\n");
        exit(1);
    }
    fdold = open(argv[1],O_RDONLY);  /* 打开源文件只读 */
    if (fdold == -1)
    {
        printf("cannot open file %s\n",argv[1]);
        exit(1);
    }
    fdnew = creat(argv[2],0666);  /* 创建可为所有用户读写的目标文件 */
    if (fdnew == -1)
    {
        printf("cannot create file %s\n",argv[2]);
        exit(1);
    }
    copy(fdold,fdnew);
    exit(0);
}

copy(old,new)
    int old,new;
{
    int count;

    while ((count = read(old,buffer,sizeof(buffer)))>0)
        write(new,buffer,count);
}
```

图 1-3 用于拷贝文件的程序

就把由名字“.”所表示的当前目录的内容拷贝到正规文件“dircontents”中；新文件中的数据与目录的内容是一个字节、一个字节地相同的，但该文件是一个正规文件（而系统调用mkond创建一个新目录）。最后我们要说的是，任一文件都可能是设备特殊文件。比如

```
copy /dev/tty terminalread
```

把在终端上键入的字符（特殊文件/dev/tty 是用户终端）读入，并把它们拷贝到文件 terminalread 上，仅当用户敲进字符 control-d 时才结束。类似地，

```
copy /dev/tty /dev/tty
```

则把终端上敲进的字符读入并把它们都拷贝回去。

1.3.2 处理环境

一个程序是一个可执行文件，而一个进程则是一个执行中的程序的实例。在 UNIX 系统上可以同时执行多个进程（这一特征有时称为多道程序设计或多道任务设计），对进程数目无逻辑上的限制，并且系统中可以同时存在一个程序（例如 copy）的多个实例。各种系统调用允许进程创建新进程、终止进程、对进程执行的阶段进行同步及控制对各种事件的反应。在进程使用系统调用的条件下，进程便互相独立地执行了。

比如，正在执行图 1-4 中的程序的进程执行系统调用 fork 以创建一个新进程。称为子（child）进程的新进程从 fork 那儿获得一个 0 返回值，并且调用 execl 以执行 copy 程序（图 1-3 中的程序）。系统调用 execl 用文件“copy”覆盖子进程的地址空间——当然，我们假设“copy”正处在当前目录中，并且，execl 根据用户提供的参数运行该程序。如果 execl 调用成功了，则它永不返回，这是因为该进程是在新地址空间中执行的，第 7 章我们将会看到这点。同时，调用 fork 的那个进程（父进程）从该调用收到一个非 0 返回值，然后调用 wait 将自己的执行挂起，直到 copy 结束时，打印出“拷贝完毕”的消息，而后退出（每个程序都是在主函数结束时退出，这是在编译处理期间与标准 C 程序库连接时由 C 程序库所安排的）。比如，若可执行程序的名字为 run，并且一个用户通过

```
run oldfile newfile
```

引用该程序的话，则进程把“oldfile”拷贝到“newfile”上并且打印出消息。虽然这个程序只往“copy”程序上添加了一点东西，但它呈现了用于进程控制的四个主要的系统调用：fork，exec，wait 及 exit。

```
main(argc,argv)
    int argc;
    char * argv[];
{
    /* 假设两个参数：源文件和目标文件 */
    if (fork() == 0)
        execl( "copy", "copy",argv[1],argv[2],0);
    wait((int *)0);
    printf( "copy done \n ");
}
```

图 1-4 创建新进程以拷贝文件的程序

一般说来，系统调用允许用户书写做复杂操作的程序，其结果是：在其他系统中成为