第一章　UNIX 操作系统概况

UNIX 是目前世界上最“古老”也是最流行的操作系统之一。它的历史悠久，而且几乎在所有计算机公司、大专院校及研究所里都可见到它的身影。回顾一下 UNIX 的产生和发展历史，可以帮助我们了解它具有如此强大生命力的原因所在，把握它的设计精神、功能特点和发展方向。

1.1　UNIX 的产生

1965 年，美国贝尔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 的新操作系统。Multics 系统的目标是为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。到 1969 年，Multics 系统已在 GE645 计算机上运行了，但是令人遗憾的是，Multics 未能达到它预期的目标，此项目最终被放弃了。当时在贝尔实验室工作的 Ken Thompson 和 Dennis Ritchie 也参加了 Multics 的工作，Multics 的工作结束后，Ken Thompson 和 Dennis Ritchie 及其他人为改善他们的程序设计环境，设想了一个新的文件系统——后来演化为 UNIX 文件系统的早期版本。Thompson 编写了若干程序，在 GE645 上模拟实现所设想的文件系统行为。与此同时，Thompson 还在 GE645 上编写了一个名为“宇宙旅行”的游戏程序，他发现此程序运行时开销太大，并且很难控制“宇宙飞船”。于是，他找来一台廉价的几乎无人问津的但却能提供良好的图形显示的 PDP-7 计算机，打算将“宇宙飞船”游戏程序移植到此机器上。移植工作并不如想象中的那样简单，Thompson 和 Ritchie 决定为 PDP-7 实现一个简单的操作系统，其中包括那个新设想的文件系统（使用层次目录结构和 inode 数据结构）支持两个用户分时共享和内存管理的进程子系统以及一部分实用程序。此操作系统是用汇编语言写的，也就是最早的 UNIX 的汇编版本。

UNIX 系统的第一位真正用户是贝尔实验室的专利部门，该部门要求在 PDP11/20 计算机上建立一个功能很强的文字处理系统。于是，第一个 UNIX 汇编版本于 1971 年移植到 PDP 11/20 上，成为 UNIX 操作系统第一版。它支持更多的用户分时共享系统资源，并带有一个功能很强的文字处理系统 roff。与此同时，Thompson 在编写 PDP 11/20 的 FORTRAN 编译器的过程中，发明了一种新的低级语言 B，在 Ritchie 的帮助下，对语言 B 进行了改造，研制出 C 语言及其编译器。C 语言以高级程序设计语言的结构和编程环境，提供了类似汇编语言那样的系统资源操纵能力和程序的执行效率。1973 年，Thompson 和 Ritchie 用 C 语言重写了 UNIX 操作系统（第四版）。重写后的 UNIX 操作系统不仅保持了原有的执行效率，而且能比较容易地移植到其他支持 C 编译器的机器上，这对整个操作系统发展来说具有重大的历史意义。

1.2 UNIX 的历史

1974 年，Thompson 和 Ritchie 在《ACM 通讯》上发表了一篇描述 UNIX 系统的文章，第一次向公众介绍 UNIX 系统，在计算机学术界引起强烈反响。贝尔实验室还将 UNIX 系统的第六版源码免费提供给一些大学、研究机构用于教学和研究，许多学者、研究人员对此表现出极大的兴趣，他们在 UNIX 系统上编写了大量的应用程序，并试图对 UNIX 系统进行改造和增强功能。例如，BSD 的第一版就是基于 UNIX 版本 6 的。到 1977 年，UNIX 系统的装机数目增长到大约 500 台，其中 20%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提供了良好的环境。这一切都使 UNIX 系统的声望日益提高。

1977 年，AT&T 开始将 UNIX 系统的许可证提供给商业机构。同年，美国交互系统公司（Interactive Systems Corporation）成了 UNIX 系统的第一个增值转卖商，他们增强了它的功能，使之用于办公室自动化环境中。 1977 年还标志着 UNIX 系统首次被移植到非 PDP 计算机——Interdata 8/32 计算机上。从那时起，许多计算机厂商纷纷向 AT&T 购买 UNIX 系统的许可证，将其移植到自己的机器上，并根据计算环境的需要作功能上的改造和增强，进而推出自己的以 UNIX 为基础的操作系统。例如，IBM 公司的 AIX 苹果公司的 A/UX、HP 公司的 HP/UX、SUN 公司的 Sun OS、Microsoft 公司的 XENIX、Intel 公司的 XENIX 286、SCO 公司的 SCO XENIX 386 等等。与此同时，AT&T 继续 UNIX 系统的研究工作，不断推出新版本。 1989 年，AT&T 的 UNIX 系统实验室（简称 USL，Unix System Laboratories）宣布了它的新版本 UNIX System V Release 4（简称为 SVR4），它将 System V、BSD 和 XENIX 统一起来。1992 年 USL 发布了 SVR4.2，它是 USL 发布的最后一个版本，同年 12 月 Novell 公司开始收购 USL 公司。1993 年底 Novell 发布了 SVR 4.2 MP，它是最后一个 System V 的 OEM 版本，Novell 公司还将 UNIX 商标和 Single Unix Specification 转给了 X/Open 公司。1995 X/Open 公司推出了 UNIX95 规范，同时 Novell 公司将 Unixware 业务卖给了 SCO 公司。1996 年 OSF 和 X/Open 合并形成 Open Group，Open Group 于 1997 年推出 Single Unix Specification 版本 2，1998 年推出 UNIX98 系列，它包括基本集、工作站、服务器三个版本。

从 UNIX 的发展历史可以看出，UNIX 系统版本较多，发展历史曲曲折折，有多种变体版本出现，一方面，表明了 UNIX 系统的普及和发展，另一方面，这些变体版本之间的不兼容给软件开发者和用户带来许多不便，造成投资的重复和浪费。因此，不管是计算机厂商还是计算机用户，都需要制定某种标准来规定应用程序设计界面（API）及其他界面，以保护各自的投资利益。目前已被广泛接受的 UNIX 标准有：

- POSIX：关于计算环境的可移植操作系统界面标准；
- SVID 由 UI（UNIX ITERNATIONAL，INC.） 组织中各成员公司联合制定的关于 SVR4 的标准；
- X/OPEN ：将国际上已公认的和事实上的标准结合起来，形成一个关于开放的交互的计算机应用环境标准 XPG；

- UNIX95 增加对实时、大文件(文件大小可以任意大)、扩展的线程功能、64 位微处理器、“2000 问题”就绪等方面的支持；
- UNIX98：UNIX95 的增强版本，兼容 UNIX95，它有基本集、工作站、服务器三个版本。

前面所述的提供类 UNIX 操作系统的公司，都宣布自己的产品符合上述一个或多个标准。

进入 80 年代后，UNIX 系统及其应用软件发展迅速，市场日益扩大，功能不断加强和完善。世界上几家主要的计算机公司都推出自己的类 UNIX 的操作系统作为主要产品或替代产品，例如 CRAY 公司为其巨型机配备的 UNICOS 和 IBM 公司的 AIX 等。迄今为止，没有一种操作系统能够像 UNIX 适应从微型机、小型机、中型机到巨型机如此广泛的硬件范围。

所有的 UNIX 操作系统可划分成三大类：

(1)基于 AT&T System V

UNIX 系统 V 是 AT&T 研制的。AT&T 拥有其版权，并向商业机构提供许可证。DEC、IBM 等大公司均购买了其许可证。

(2)基于 BSD

在美国国防部的资助下，由加州大学伯克利分校的一个研究小组研制的一个 UNIX 系统，称作 BSD。目前，UNIX 系统中广泛采用的虚存管理、全屏幕编辑、Socket 机制和网络功能等首先是在 BSD 中实现的。如果说 AT&T UNIX 适合商业事务性处理，那么 BSD 更适合于软件研究开发。目前 BSD 最高版本是 4.4 版，1993 年推出 BSD4.4 之后，由于 Novell 公司收购了 USL，为防止侵犯 USL/Novell 公司版权，BSD 于 1994 年推出了 4.4-lite，删除了所有与之相关的代码。

(3)基于 XENIX

XENIX 是 Microsoft 公司为 Intel 一族的微型机研制的类 UNIX 系统。SCO 公司购买了其版权，研制出自己的 XENIX 386。1995 年 SCO 公司从 Novell 公司购买了 Unixware 的业务，并在此基础上不断改进，于 1997 年推出了 Unixware 7。

1.3 UNIX 的特点

如果用一句话来描述 UNIX 系统的特点，那就是功能简洁、层次清晰。这主要是由最初研制它的人员和环境所导致的。Thompson 和 Ritchie 研制这个系统时不是为开发商品，是想为自己建立一个舒适的软件开发环境。因此，设计时力求简单实用，这样设计出来的操作系统就非常简洁。由此带来的结果是，UNIX 系统具有许多引人注目的特点：

- 该系统以高级语言书写，易读、易懂、易修改，易移植到其他机器上。据 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，大约慢 20%～40%。但是，采用高级语言的优点要多于它的缺点。现在研制的操作系统大都采用 C 语言编写程序；
- 它提供简单的基本的系统调用界面和一个功能很强的命令处理器 shell，支持一组

能够由较简单的程序构造出复杂程序的原语，如管道操作、I/O 改向。这使得软件开发人员能够方便地编写高效率的应用程序。 UNIX 系统的许多高级功能都是以实用程序的形式实现的；

- 它使用了易于维护、实现高效的层次结构文件系统。文件采用字符流这样的一致格式，使应用程序易于编程和处理；
- 它是一个多用户、多进程系统。目前已开发出 UNIX 多机、多线程版本。
- 支持多种文件系统，例如，s5、ufs 等；
- 它是一个开放系统，已有公认的 API 标准。用户可以方便地编写 UNIX 应用程序，而不必考虑具体的机器硬件结构。

UNIX 系统也有一些不足之处，例如 UNIX 的用户界面不太友好。对一般用户而言它有点难以掌握，它更适合软件开发人员的口味。现在，UNIX 系统上的图形界面系统正努力弥补这一不足之处。 UNIX 不太适合实时处理，因为它只能在预定的几个“安全点”作进程切换来响应实时处理。

AT&T 的 UNIX 系统 V SVR4 中实现的主要功能有：

TCP/IP 协议；
C shell 、kern shell；
套接字(sockets)机制；
作业控制；
符号链接；
增强的信号处理；
Berkeley 快速文件系统 (UFS)；
虚拟文件系统(VFS)；
远程文件系统(RFS)；
进程间通信(IPC)；
远程过程调用(RPC)；
vnodes 数据结构；
Open look 窗口系统；
内存映射文件；
外部数据表示(XDR)；
共享库；
流(STREAMS)通信机制；
实时支持。

1.4 UNIX 的未来发展

目前，UNIX 系统发展正进入一个新的重要时期。新的软件技术、网络技术及安全技术不断引进 UNIX 系统，UNIX 系统及其变体的新版本纷纷出台，令人眼花缭乱。例如，USL 与 Novell 公司共同开发的 SVR4 的桌面系统版本 UnixWare。Unixware 具有支持更

强的网络能力，提供更友好的用户窗口界面。

UNIX 虽然是一个易移植、易扩充的系统，但是，多年来人们不断地在 UNIX 系统内核中添加新的功能和特性，以满足计算环境的需要，使原来那个比较小、比较紧凑的内核逐渐变得庞大而复杂，难以维护。越来越多的人开始意识到这一点，并寻求解决问题的途径。1984年卡内基梅隆大学采用微内核(Microkernel)技术研制出一个新的操作系统核心 Mach 就是一个成功的范例。

Mach 的设计思想是，核心只提供必要的几个简单的用于建立复杂操作的对象，将尽量多的传统操作系统的功能模块放到核心外的用户层实现，而这一点满足了 UNIX 的初衷 —— 小而精。在 Mach 3.0 中，UNIX 是作为核心外的一个应用程序 ——UNIX server 的身份实现的，可提供与 BSD4.3 完全二进制兼容的应用环境。Mach 具有良好的可移植性、可调性和可扩充性，可支持松散型及紧密藕合型多处理机体系结构，提供良好的分布及并行计算环境。

OSF(Open Software Fundation)在它推出的 OSF/1 操作系统中采用了 Mach 作为核心。OSF/1 在安全性、多处理、多线程及其他特性都有所加强的 Mach 基础上，提供 BSD 的内核功能，能很好地支持多处理环境和多线程并发，并遵从所有有关的 UNIX 标准和规范，Compaq 公司的 Tru64 UNIX 就是采用 Mach 内核。UI(UNIX International) 也表示，将在它的新 UNIX 多机版本中吸收 Mach 的优点及采用微内核技术。

近几年来，随着 Internet 的飞速发展，一种 UNIX 的克隆操作系统 Linux 越来越引起大家的关注。Linux 的出现和发展有其历史原因，一是通常的 UNIX 版本在低端计算机上费用较高，如当时在 386 上运行 AT&T 的 System V 需要 1500 美金；二是 Internet 的发展为软件开发人员共同合作提供了较好和较快的交流手段。

Linux 是免费软件，任何人都可以从 Internet 上得到它的源代码，在安装使用时不需要花费较多的资金，它与 UNIX 有完全不同的运作模式。 1991 年 10 月 5 日，Linus Torvalds 公布了第一个 Linux 版本 0.02，当时 Linux 只能运行 bash 和 gcc。继版本 0.03 之后，越来越多的人参与了 Linux 操作系统的开发，可以说 Linux 是所有参加者共同的智慧结晶。目前，Linux 可以运行 X Windows、TCP/IP、Emacs、UUCP、mail 和 news 软件、数据库软件等，Linux 不仅仅运行在 Intel 平台上，也可以运行在 Alpha、MIPS 等高性能微处理器上。Linux 为操作系统的发展带来了广阔的前途和机遇。

第二章　UNIX 操作系统原理

2.1　基本概念

概括地说，操作系统主要有两方面的功能。一是给用户提供一个方便而且强有力的使用计算机资源的环境，二是充分合理地分配使用系统内包含的各种软硬件资源，提高整个系统的使用效率和经济效益。在操作系统的支持下，用户可以访问远程文件而不必关心网络操作细节，用户可以分时共享 CPU 而不会感觉到他人的使用。因此，任何操作系统都是以资源的管理、共享和使用为核心的。

计算机资源大体上可分为硬资源和软资源。硬资源包括存储器、CPU和各种硬件设备，软资源包括程序、数据、地址空间等。实际上，没有绝对的软、硬资源之分。因为，任何硬件资源都需要用软件数据结构来表示，而任何软件资源都必需依附某种硬件资源。例如，I/O 设备用设备文件来表示；所有程序和数据都需要存放在内存或磁盘。

在操作系统中，每个用户都是以注册名或用户名代表其身份的。用户的每个命令和程序都是以进程的形式在系统中运行的，而用户的程序和数据则是以文件的形式存放在磁盘中。因此，文件和进程是 UNIX 系统中两个最基本的概念。

文件是具有名字的有序的数据集合。例如，一个 C 或 FORTRAN 源程序、一个目标代码程序、系统中的库程序、一批待加工处理的数据、一篇文章等都可构成一个文件。进程是程序的一次执行，是系统进行资源分配和调度运行的一个独立单位。一个进程循着一个严格的指令序列执行，这个指令序列是自包含的，并且不会跳转到别的进程的指令序列上。它读写自己的数据和栈区，但不能读写其他进程的数据和栈区。进程可以通过系统调用与其他进程及外界进行通信。核心为每个进程建立一张资源登记表，记录它使用的各种资源，如内存、文件等。

UNIX 系统的体系结构示于图 2.1。UNIX 核心直接与硬件交互，向外提供 UNIX 系统调用界面。外层的程序，诸如 shell 及编辑程序(ed 与 vi)，通过引用系统调用请求核心完成各种操作，并在核心与调用程序之间交换数据。其他应用程序能在较低层的程序或工具的基础上构筑而成，因此它们存在于图的最外层。虽然图 2.1 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，把现存程序组合起来，完成更复杂的任务。

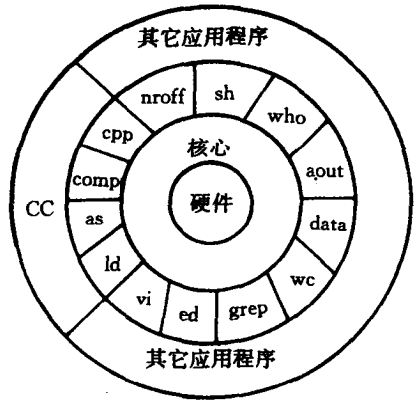

图 2.1　UNIX 系统的体系结构

图 2.2 给出了 UNIX 核心框图，示出了各

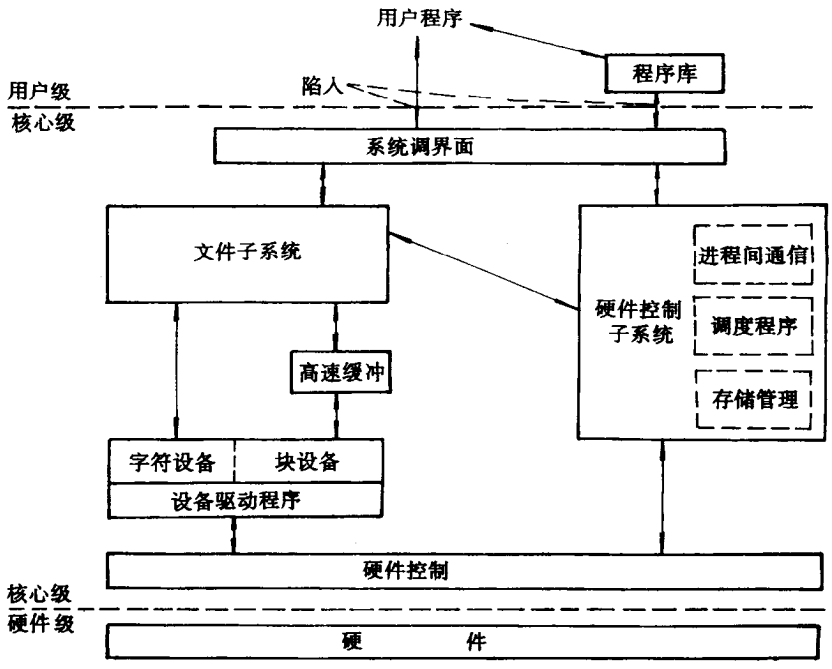

图 2.2　系统 UNIX 核心框图

种模块及它们之间的相互关系，特别是它示出了核心的两个主要成份：文件子系统和进程控制子系统。图 2.2 让我们看到了三个层次：用户、核心及硬件。系统调用与库接口体现了图 2.1 中描绘的用户程序与核心间的边界。系统调用看起来像 C 程序中普通的函数调用，而库把这些函数调用映射成进入操作系统所需要的原语(例如 trap 指令序列)。不过，汇编语言程序可以不经过系统调用库而直接引用系统调用。像标准 I/O 库这样一些其他的库程序，可以提供用户对系统调用的更高级的使用。

图 2.2把系统调用的集合分成与文件子系统交互作用的部分及与进程控制子系统交互作用的部分。文件子系统管理文件，包括分配文件空间、管理空闲空间、控制对文件的存取以及为用户检索有关文件目录。文件子系统使用一个缓冲机制存取文件数据，缓冲机制调节在核心与磁盘之间的数据流。缓冲机制同块 I/O 设备驱动程序交互作用，以便控制数据在核心与设备之间的传输。文件子系统在没有缓冲机制的干预下，直接与字符设备（包括所有非块设备）驱动程序交互作用。任何一个 I/O 设备都作为文件系统中的一个特殊文件加以管理和控制，用户可以像操作普通文件一样操作某个设备。

进程控制子系统负责进程同步、进程间通信、存储管理及进程调度。当要执行一个文件而把该文件装入存储器中时，文件子系统与进程控制子系统交往。进程子系统在执行可执行文件之前，先要为它分配存储空间，由文件子系统将文件读到主存。存储管理模块控制存储分配。在任何时刻，只要系统没有足够的物理存储空间供进程使用，核心就在主存与磁盘之间对进程进行迁移，以便所有的进程都得到公平的执行机会。调度程序模块把 CPU 分配给进程。该模块调度各进程依次运行，直到它们因等待资源而自愿放弃 CPU，

或直到它们的运行时间超出一个时间片。这时，调度程序选择最高优先权的就绪进程投入运行，当原来的进程成为最高优先权的就绪进程时，还会再次投入运行。进程间通信模块负责进程之间的事件信号的传达和消息的传递。

硬件控制模块负责处理中断及与机器通信。像磁盘或终端这样的设备可以在一个进程正在执行时中断CPU。如果出现这种情况，在对中断服务完毕之后，核心可以恢复被中断了的进程的执行。中断不是由特殊的进程服务的，而是由核心中的特殊函数服务的，这些特殊函数是在当前运行的进程的上下文中被调用的。

2.2 进程管理

2.2.1 进程的概念

如前所述，进程是系统进行资源分配和调度运行的一个独立单位。一个进程有自己独立的虚地址空间、核心栈和运行现场（进程上下文）。进程从创建到消亡的过程称为进程的生存周期。所谓进程状态是指进程的生存状态：是运行还是挂起；进程的执行态是指进程运行的硬件环境：用户态或核心态。

用户态与核心态之间的区别是：用户态下的进程只能访问用户虚空间，核心态下的进程可以访问核心虚空间和用户虚空间。硬件将一个进程的虚地址空间划分成仅在核心态下可存取和在核心态、用户态下都可存取的两部分；某些机器指令是只能在核心态下执行的特权指令，在用户态下执行特权指令会引起错误。例如，一条操作处理机状态寄存器的指令在用户态下执行的进程没有执行特权指令的能力。

当一个用户进程执行一个系统调用时，进程的执行态从用户态变为核心态，即由操作系统执行并为用户请求服务。显然，系统在运行时必处于核心态和用户态之一，但核心是为着用户进程运行的，核心并不是与用户进程平行运行的孤立的进程集合，而是每个用户进程的一部分。当然，核心也会创建一些进程（称为核心进程）来运行专门的核心管理程序，如页面对换进程、空闲(idle)进程、优先数动态计算进程等，这些核心进程通常只在核心态下运行。所谓‘核心’分配资源或‘核心’进行某种操作的真实含义是，一个在核心态下执行的进程执行分配资源或进行某种操作的操作系统代码。例如，shell通过系统调用读用户终端，shell进程进入核心态，执行有关的核心代码对终端的操作进行控制，返回读入字符，然后shell进程回到用户态下运行，对用户敲进来的字符流进行解释，执行特定的动作集合，在这些动作集合中又可以引用其他系统调用。

在UNIX中，描述一个进程的核心数据结构是：进程表项(proc结构)和u区(user结构)。

进程表项是在核心虚空间中分配的，因此对核心来说总是可存取的。它含有：

- 进程标识：唯一地标识本进程；
- 父进程标识：记录父进程的进程标识；
- 运行状态：记录进程当前的状态。进程的运行状态通常有：运行（runing）、就绪（ready）、挂起或睡眠(suspernd)、换出(swapped)；

- 进程上下文：进程的运行现场（包括所有寄存器的值，如程序寄存器、处理机状态寄存器等）。当进程“放弃”或“被抢占”CPU时，核心将进程的运行现场保存在此域中，以便下次调度该进程时，恢复其运行现场；
- 本进程区表：用于管理和分配进程的虚地址空间；
- 软中断信号域：记录发向一个进程的所有未处理的软中断信号；
- 其他：包括调度参数、计时域等。

u区分配在用户的虚空间中，因此只能由本进程来存取。每当创建一个进程时，核心都将为其分配u区。它含有：

- proc指针：指向对应于该u区的进程表项；
- 用户标识号：决定了进程的各种权限，如文件存取权限；
- 计时器域：记录进程(及其子进程)在用户态和核心态运行时所用的CPU时间；
- 信号处理表：记录进程对各种信号的处理方法；
- 控制终端域：标识与进程相关的“注册终端”(如果存在的话)；
- 核心返回值域：记录系统调用的返回值和遇到的错误；
- 当前目录和当前根：描述进程的文件系统环境；
- 用户文件描述符表：记录该进程已打开的文件；
- 其他：包括其他的资源登记和使用权限。

虽然，一个进程的虚空间包括核心虚空间和用户虚空间，通常所称的进程虚空间是指进程的用户虚空间。

2.2.2 进程上下文及切换

进程的上下文是由它的用户级上下文(user-level-context)、寄存器上下文(register-context)以及系统级上下文(System-level-context)组成。

用户级上下文是由进程的正文(程序)段、数据段、用户栈和共享存储区组成。它们占据了该进程的虚地址空间。由于对换操作或调页，进程虚地址空间的某些部分不总是驻留在内存中，但它们仍然是用户级上下文的组成部分。

寄存器上下文由下列部分组成：

- 程序寄存器(或简称为PC)。其中指出CPU将要执行的下条指令的地址，该地址是核心空间或用户空间的虚地址；
- 状态寄存器(或简称为PS)。其中给出处理机与该进程相关联时的硬件状态，例如，发生例外或中断的标志位，进程的执行态(用户态或核心态)标志；
- 栈指针：含有栈中下一项的地址。由进程的执行态来决定该地址是指向核心还是指向用户栈；
- 通用寄存器：其中的数据是进程在其运行期间产生的。在进程和核心之间传送信息时要用到它们。

进程的系统上下文包含：

- proc表项：它记录了该进程的状态，并含有核心总能取到的控制信息；
- u区：其中含有进程的控制信息，这些信息只需在该进程的上下文中被存取；

- 虚空间分配表：即本进程区表表项、区表及页表。它们定义了从虚地址到物理地址的映射，因而决定了进程的正文区、数据区、栈区和其他的区；
- 核心栈：进程在核心态下执行时使用的堆栈。尽管所有的进程执行相同核心代码，但它们各有自己的核心栈。

当发生中断时，或一个进程发生系统调用或进程进行上下文切换时，核心就在当前进程的核心栈中压入一个寄存器上下文层。当核心从处理中断中返回，或一个进程在完成其系统调用后返回用户态或一个进程进行上下文切换时，核心就从当前进程的核心栈中弹出一个寄存器上下文层。因此，进程上下文的切换总会引起一层寄存器上下文的压入或弹出，即核心压入老的进程的上下文层，弹出新的进程的上下文层，然后开始或继续新的进程的执行。进程表表项中存放着当前上下文压所必要的信息。图 2.3示出了进程切换的过程。

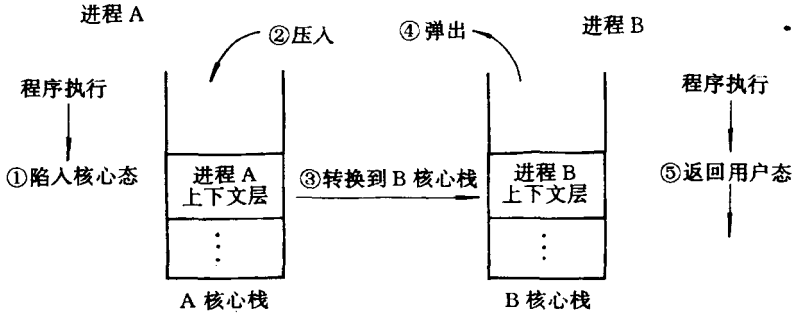

图 2.3 进程 A 与 B的上下文切换过程

UNIX 系统允许 I/O 外围设备或系统时钟异步地中断 CPU。核心处理中断的操作顺序是：

1)在当前进程核心栈中保存（压入）上下文层，同时创建中断处理所需的上下文层（给寄存器赋初值）。

2)确定中断源,识别中断类型(如时钟或磁盘)。核心通常把同级或较低级的中断屏蔽掉，仅允许较高级的中断，这样做的目的是避免因竞争造成核心数据的不确定，同时能及时地响应高级中断。

3)核心根据中断号查中断向量表（含有每种中断源的中断处理程序地址以及中断处理程序取得参数的方式），调用中断处理程序。可以用当前进程的核心栈存放中断处理程序的栈结构，也可以使用全局中断栈存放中断处理程序的栈结构。后者能保证中断处理程序不用进行上下文切换就能返回。

4)中断处理程序工作完毕并返回。核心执行一系列特殊机器指令，恢复到中断前的核心栈和上下文层，被中断的进程继续执行，就好像什么事也没发生过一样。

例外事件指的是由一个进程自身引进的非期望事件，例如非法存储器寻址、执行特权指令、除数为零等等，它们与来自进程外部的事件所引起的中断是有区别的。例外事件发生在一条指令执行的过程中间，并且系统试图在处理完例外事件之后重新启动该指令，中断被认为是在两条指令的执行之间发生的，系统在对中断服务完毕之后，从下一条指令继

续执行。核心对例外事件的处理过程基本上与中断处理相同。

2.2.3 进程的状态与控制

一个进程的生命周期从概念上可分为一组状态，这些状态刻划了进程从创建、活动到消亡的过程。下面列出了 UNIX 进程状态的完整集合。

- 进程在用户态下正在执行；
- 进程在核心态下正在执行；
- 进程没有被执行，但处于就绪状态，只要核心调度到它，即可执行；
- 进程正在睡眠并驻留在主存中；
- 进程处于就绪状态，但驻留在磁盘上，等待对换进程将它换入主存；
- 进程正在睡眠且驻留在磁盘上；
- 进程正从核心态返回用户态时，核心抢先于它，并做了上下文切换，以调度另一个进程；
- 进程刚被创建；
- 进程执行了系统调用 exit，处于僵死(zombic)状态。它是进程的最后状态。

图 2.4 给出了完整的进程状态转换图。下面让我们看一个典型的进程经历这个状态转换模型的过程。首先，当父进程执行系统调用 fork 创建子进程时，新创建的进程进入状态模型中的“创建”状态，并最终会移到“在内存中就绪”状态。当进程调度程序选取这个进程运行时，它便进入“核心态运行”状态。当该进程完成系统调用 fork 时，它进入“用户态运行”状态，此时它在用户态下运行。一段时间后，外部可能中断处理机，例如，接到某个外设的中断信号或接到时钟中断信号，进程则再次进入“核心态运行”状态。当中断处理程序结束时，核心可能决定调度另一个进程运行。这样，头一个进程就进入“被抢先”状态，而后者开始了它的运行。当调度程序再次选取我们例子中的进程去执行时，它便回到“用户态运行”状态继续运行。

假定进程执行读/写文件的系统调用，它此时便离开“用户态运行”状态，进入“核心态运行”状态。如果该进程需要等待磁盘输入/输出的完成，则进入“在内存中睡眠”状态，一直睡到被告知输入/输出已完成。当输入/输出完成时，硬件便中断 CPU，中断处理程序唤醒该进程，使它进入“在内存中就绪”状态，等待被重新调度运行。

假定系统中有多个进程同时运行，但它们不能同时都装入主存，核心对换进程决定换出我们的进程，以便为另一个处于“就绪且换出”状态的进程腾出空间。当进程被从主存中驱逐出去后，它进入“就绪且换出”状态。当以后对换进程选择我们的进程换入主存时，它便重新进入“在内存且就绪”状态。最后，当该进程完成时，它发出系统调用 exit，进入“核心态运行”状态，最终进入“僵死”状态。

一个进程可以发出系统调用，实现从“用户态运行”状态到“核心态运行”状态的状态转换。但是，进程不能控制何时从核心返回。有些事件（例如软中断信号）可能强迫它永不返回而进入“僵死”状态。

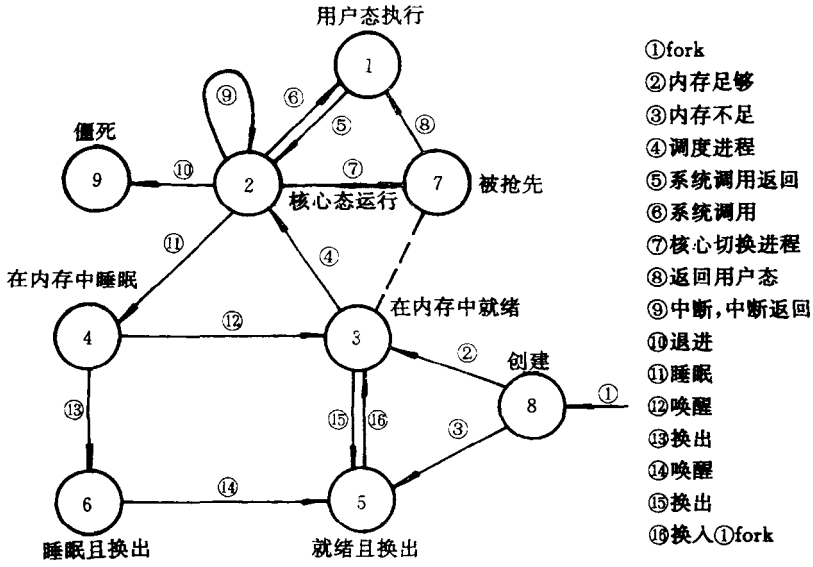

图 2.4 UNIX 进程状态转换图

2.2.4 进程的调度

UNIX 系统的进程调度程序采用的是多级反馈循环调度方法。核心扫描所有就绪队列，选择已装入内存、优先权最高且“就绪”时间最长的进程投入运行，并赋给进程一个 CPU 时间片。当该进程因等待系统资源进入睡眠，或者运行完其时间片时，核心将它反馈到若干优先级队列中的某一个队列，并调度下一个“合格”进程运行。如果没有合格的进程，核心则运行休闲(idle)程序。一个进程在它结束之前，可能需要多次通过“反馈循环”。当核心做进程切换和恢复一个进程的上下文时，该进程从它原来被挂起的地方继续执行，直到下次中断发生（最迟到下一次时钟中断的发生）。在处理完中断后，核心再次运行调度程序。

每个进程都有一个优先域，其值称为优先数。优先数愈低，优先权愈高。每个优先数都有一个进程队列(见图 2.5)，每个就绪进程都挂在其中的一个队列里。图 2.5 中示出进程优先权的范围分为两种：用户优先权和核心优先权。其中的一个优先权阈值作为核心优先权与用户优先权的分水岭。核心优先权又可进一步划分为不可中断优先权和可中断优先权。这里的“中断”是指中断进程的睡眠过程。进程在进入系统调用后会因为等待资源而睡眠，这时如果收到一个软中断信号，具有可中断优先权的进程可被唤醒，也就是说，可“中断”本次系统调用或资源等待，而具有不可中断优先权的进程却继续睡眠。

在 UNIX 系统中，进程的优先权是动态变化的。核心根据以下原则和进程的运行状态计算一个进程的优先权。

核心将特定的优先权赋给一个即将进入睡眠的进程，也就是说，将一个固定的优先数和睡眠的原因（即等待的资源对象）联系起来。目的是使那些可能缓减系统资源瓶颈的进程得到较高的优先权。举例说来，一个睡眠等待磁盘 I/O 的进程比等待一个自由缓冲区的

进程具有较高的优先权。这是因为，等待磁盘 I/O 完成的进程已经有了缓冲区，当它醒来时，它就有机会做足够的处理，从而释放该缓冲区以及可能的其他资源。它释放的资源越多，其他进程运行的机会就越多，不用挤在那里等待资源。

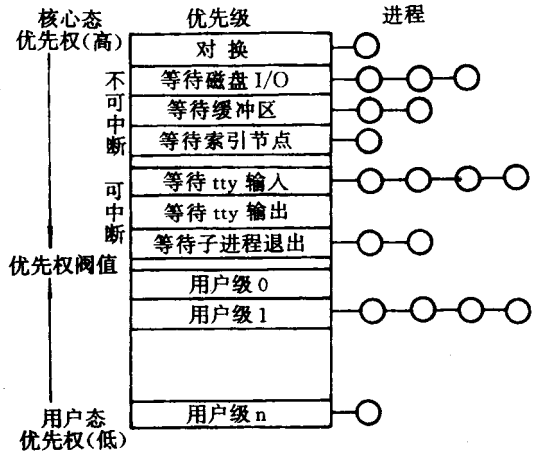

图 2.5 进程优先权范围

核心调整从核心态返回到用户态的进程的优先权。该进程以前可能经历了睡眠状态，其优先权已变到一个核心优先权，因此必须在返回用户态时被降低到用户级优先权。同时,为对其他进程公平起见,核心要处罚该进程,因为它刚刚占用过宝贵的核心资源。

时钟处理程序以固定的间隔（通常为 1 秒）调整用户态下的所有进程的优先权，使优先权成为进程最近使用 CPU 时间的函数。最近使用较多 CPU 时间的进程得到较低的优先权,以防止某个进程垄断 CPU 的使用。

固定间隔地重新计算优先权值的效果是，具有用户级优先权的进程在优先级队列之间移动，最终成为当前最高优先权者被调度运行。核心并不改变核心态进程的优先权，也不允许具有用户级优先权的进程跨越优先权阈值而获得核心级优先权，除非这些进程进行系统调用并进入睡眠。

当核心唤醒一个进程时,或当时钟处理程序修改所有‘ 就绪 ’进程的优先权时,可能会出现一个比当前运行进程具有更高优先权的进程，或者当时钟处理程序确定正在运行的进程已经用尽了它的时间片，核心将运行进程调度程序来重新调度一个进程。

2.2.5 软中断信号

软中断信号是 UNIX 进程控制中的一个重要概念，主要用于通知进程发生了异常事件和进程跟踪机制。进程之间可以通过系统调用 kill相互发送软中断信号,核心也可以从内部向进程发送软中断信号。UNIX 的软中断信号可分成下列几类：

- 与进程终止相关的软中断信号。当进程自行退出或被要求退出时，发送这类软中断信号；
- 与进程例外事件相关的软中断信号。如进程访问一个在其虚地址空间以外的地址，

或企图写向一个只读内存区（如程序正文区），或执行一个特权指令及其他各种硬件错误；

- 由执行一个系统调用时遇到的非预测错误条件所引起的软中断信号。例如，调用一个不存在的系统调用；向一个没有接收进程的管道线写数据；
- 由用户态下的进程发出的软中断信号。例如，一个进程想在一段时间后收到一个闹钟信号,或进程使用系统调用 kill 向其他进程发送任意的中断信号；
- 和终端交互有关的软中断信号。例如，当用户按了终端键盘上的“ break ”键或“delete”键时所产生的软中断信号。

进程通过系统调用 signal 来规定收到某种信号时自己要做的动作：或退出（exit）;或忽略信号，不做任何处理；或执行一个特殊的用户处理函数。缺省动作是调用 exit。核心将用户指定的处理函数地址存放于进程 u 区中的软中断信号处理表中。

进程通过系统调用 kill 向指定的进程发送一个软中断信号，核心在被指定进程的进程表项中的软中断信号域里置上相应位，表示该进程已收到此信号。如果该进程睡眠在一个可被中断的优先权上，核心就唤醒它。

当一个进程即将从核心态返回到用户态时，或它要进入或者离开一个适当的低调度优先级睡眠状态时，核心要检查它是否收到一个软中断信号。核心仅当一个进程从核心态返回用户态时才处理软中断信号。因此，当一个进程在核心态下运行时，软中断信号并不立即起作用。如果一个进程正在用户态下运行，而且核心处理一个使某个软中断信号发送给该进程的中断，那么，当核心从中断返回后，它将识别和处理该软中断信号。

在处理软中断信号时，核心首先确定信号的类型并清除进程表项中相应的信号位。如果软中断信号处理函数被置为缺省值或退出 (exit)时,核心在 exit 之前要转贮进程的‘ 内存 ’映像(core)，以便用户确定转贮的原因并可以调试他们的程序。如果是被忽略的信号，核心使进程返回用户态，像没有收到该信号似地继续执行。如果是要捕俘的信号，核心使进程在返回用户态后，立即执行用户定义的（在 u 区的软中断信号处理表中登记的)软中断处理函数，其具体步骤为：

1)核心存取用户保存的寄存器上下文（在用户栈中),找出其中的程序计数器和栈指针。

2)清除 u 区中相应的软中断信号处理函数域，将其置为缺省值。

3)核心在用户栈上创建一个新的栈层，写入步骤 1)中取出的程序计数器和栈指针的值，就好像进程调用了一个用户层函数（软中断信号处理程序),调用点就是进程作系统调用或核心中断它的地方（识别信号之前).

4)核心改变用户保存的寄存器上下文，将程序计数器置为软中断信号处理函数的入口地址，并将栈指针置为用户栈增长后的值。

在从核心态返回用户态后，进程将执行软中断信号处理函数。当从软中断处理函数返回时，进程回到系统调用或中断发生时的用户代码处，就像是从系统调用或中断返回一样。

2.2.6 系统自举和进程树

在 UNIX 系统中,自举过程最终要读一个自举程序,并将其装入内存执行。自举程序可以存放在磁盘的自举块（第 0 块)中，也可以存放在只读存储器中。自举程序的主要功能是将 UNIX 核心程序从文件系统（例如从文件“ /UNIX”)中装入内存。在核心装入内存后，自举程序将控制转到核心的起始地址，核心开始运行。

核心首先初始化它的内部数据结构。例如，构造空闲缓冲区和索引节点的链表，构造缓冲区和索引节点的散列队列，初始化区表、页表等等。初始化工作完成后，核心将根文件系统安装到根(“/”),自己成为 0＃进程,并创建 1＃进程和其他进程。

然后,1＃进程又为每个终端生成一子进程(P_{20}、P_{21}、…、P_{2m}) 。这些子进程等待用户登记(login)使用系统,接着执行 shell 命令解释程序。在此过程中，它们又可能各自创建若干子进程，每个子进程执行一条命令。执行 shell 命令的子进程也可以按需要再创建子进程。依此类推,UNIX 系统中的进程构成了树形结构的进程族（见图 2.6)。除了系统中同时存在的进程数受到限制外，树形结构的层次可以不断延伸。

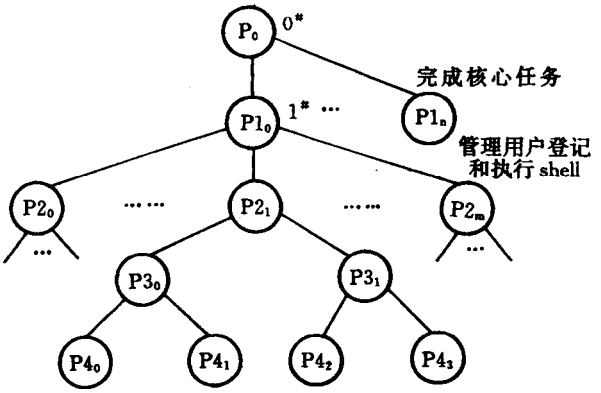

图 2.6 UNIX进程的树形体系

2.3 存储管理

2.3.1 基本概念

所有存储器（包括只读存储器和随机存储器)按照线性地址进行编址，成为一个整体，它们的地址集合称为物理地址空间或物理存储空间。其中不仅要存放操作系统的程序和数据，还要存放用户的程序和数据。当程序和数据被编译时，编译程序从零地址开始为程序和数据编排地址。这些地址的集合称为程序地址空间。

前面已经提到,程序都是以进程形式执行的,因此,光有程序和数据是不够的,还必需有与进程相关的其他数据。当执行一个可执行文件时，操作系统需要创建一个进程，并把可执行文件中的程序、数据与进程本身的其他数据合并成一个地址空间。这个地址空间就

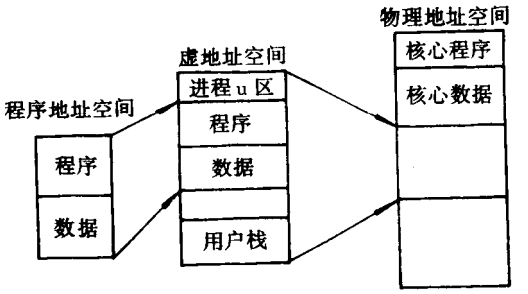

图 2.7 三种地址空间的关系

是程序执行过程中使用的地址空间，称为虚地址空间。通常为了提高执行效率，由编译程序采用预留地址的方法，在生成一个可执行文件时完成虚地址空间的装配工作。

图 2.7 示出了上述三种地址空间之间的关系。通常，物理存储空间的低地址部分用于存放 UNIX 核心程序和数据，高地址部分用作 UNIX 的高速缓冲区，只有中间部分才可以分配给用户使用。

每个进程拥有自己的虚地址空间，所有进程分享使用系统的物理存储空间（实空间）。存储管理模块的功能就是：管理和分配进程的虚空间以及系统的实空间；负责虚、实空间的映射变换；将虚存地址转换成机器能识别的物理存储地址。

2.3.1 页式(paging)存储管理

UNIX 系统采用页式存储管理技术。核心仅当进程需要时，才为进程分配物理存储页，这一过程称为请求调页。请求调页系统将进程从物理存储器的大小限制中解放出来，除了虚空间大小的限制之外，请求调页对用户是透明的。

进程常表现为在一段时间仅执行一部分指令和访问一部分数据空间，这种现象被称为“局部化原理”。我们把进程某段时间内访问的页面集合称为进程在这段时间上的工作集。核心采用某种页面交换算法，尽量使工作集中的页面驻留在内存，而将不在工作集中的页面换出到磁盘中。

页式存储管理中的主要核心数据结构是页表。每张页表描述了一个虚空间与实空间的映射关系。核心将虚地址空间和物理地址空间划分成大小相等的页面，每个虚页面在页表中占一项。其中有：

(1)物理页面描述项

物理页面地址；

页面保护位(读、写、执行)；

页面状态位(有效位、访问位、修改位、写时拷贝位等)。

(2)磁盘块描述项

设备号；

块号；

类型(对换、文件、清零、填入)。

以虚页地址为索引读取页表项就可得到此虚页对应的物理页地址或此虚页在磁盘中的块地址。

实现页式存储管理的主要硬件机制是快速联想寄存器（又称为快表）和地址变换与机制。快表具有并行查找能力，用于存放虚、实页面地址映射项。每个映射项含有：虚页号和物理页面描述项（与页表中相同）。

当进程使用虚地址访问指令或数据时，硬件从此虚地址中截取虚页号，将此虚页号与快表中的每个映射项的虚页号相比较，取得符合者的物理页地址。将物理页地址与虚地址中的页内偏移相加，形成物理存储地址，访问存储器。整个地址变换过程如图 2.8所示。

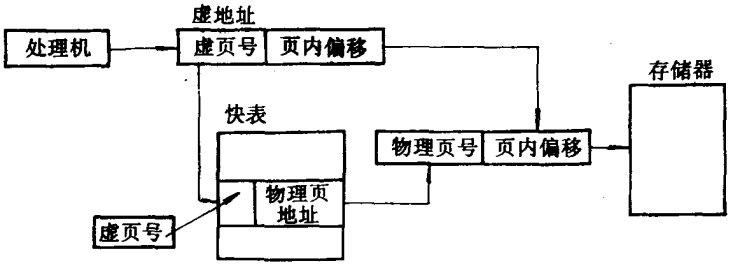

图 2.8　页式系统地址变换过程

由于快表容量有限，只能存放有限的映射项。当硬件发现某个虚页号在快表中无符合项时，发生页面失效例外。核心进入页面中断处理，它将根据虚地址查找相应的页表表项。如果表项中有效位为 0，或者页面保护位不允许此次访问，核心将报给进程‘访问越界’错误信息。如果此页面已在内存中，核心就形成虚、实页面映射项，写入快表中（可能需要替换快表中的一个映射项），然后从中断返回，重新执行进程的访存指令。如果此页面不在内存中，核心将挂起进程，同时唤醒页面对换进程，请求换入此页面。

页面对换进程负责页面的换入/换出工作。它查看所有物理页面的状态，将那些最近未使用的、或最久未使用的页面写到磁盘上，同时，根据页表中的磁盘块描述项把需换入的页面读到内存中来，并唤醒等待页面换入的进程。

页表项中还包含一些控制位，通过这些控制位可以实现物理页面的共享。在一个进程试图写一个页面，而该页面的页表项中置有‘禁止写’或‘写时拷贝’位时，硬件将发生页面保护例外。核心调用页面保护例外处理程序，根据例外发生时的虚地址找到相应的区和页表表项。如果是‘禁止写’，核心返回给进程‘访存越界’的错误信息 如果是‘写时拷贝’，核心则分配一个新页，将旧页的内容拷贝到新页上并修改进程相应的页表项。例外处理返回后，进程将写入新页，此后也只能访问新页，而其他共享此页的进程仍然保持引用旧的页面。

2.3.2　进程虚空间描述

在 UNIX 系统 V 中，一个进程的虚地址空间可以分成若干逻辑区(region)。区是进程虚地址空间上的一段连续区域，也是可被共享和保护的独立实体。一个进程通常有正文(text)、数据(data、bss)及栈三个独立区。若干进程可以共享一个区。例如，几个进程可以