北京希望电脑公司UNIX SYSTEM V4.0技术丛书

UNIX SYSTEM V
操作系统内核代码剖析

樊建平　王永杰　刘文卓
战　超　刘　宏　孙培兴　编写

樊建平　王永杰　刘晓华　审校

海洋出版社

UNIX SYSTEM V
操作系统内核代码剖析

樊建平　王永杰　刘文卓
战　超　刘　宏　孙培兴　　编写

樊建平　王永杰　刘晓华　　审校

海　洋　出　版　社

1992 年 4 月

内 容 摘 要

本书全面系统地介绍了 UNIX 系统的核心结构，对核心的源代码以及系统初启的过程作了较为深入的剖析。本书涉及 UNIX 核心中的各个方面，内容详尽，行文深入浅出，具有很强的可读性。

本书可作为大学计算机科学系高年级学生和研究生的教学参考书，也可供广大的 UNIX 设计和研究人员及实用程序开发人员参考。

欲购本书的用户可直接与北京 8721 信箱联系，邮码 100080，电话 2562329

(京)新登字 087 号

* *

责任编辑 阎世尊

北京希望电脑公司 UNIX SYSTEM 技术丛书

UNIX SYSTEM V

操作系统内核代码深入剖析

樊建平 王永杰 刘文卓

编写

战 超 刘 宏 孙培兴

审校 樊建平 王永杰 刘晓华

* *

海洋出版社出版（北京市复兴门外大街 1 号）

海洋出版社发行兰空印刷厂印刷

开本：787 x 1092 1/16 印张 23.125 字数：557 千字

1992 年 5 月第一版 1992 年 5 月第一次印刷

印数 1 — — 4000 册

ISBN 7—5027—2415—X/TP. 67

定价： 15.00 元

目　　录

第一章　系统概述

1.1　UNIX 概述

UNIX 是一个操作系统，是一组控制用户程序与最低层的机器功能交互的程序。UNIX 操作系统控制计算机的资源，使多个用户可以并发访问这些资源。UNIX 负责任务（进程）调度，控制与系统相连的外设，提供文件系统管理功能，并对终端用户屏蔽内部机器结构。

UNIX 还是一个程序设计环境，它为程序员提供了丰富的软件开发工具，以工具箱的形式对软件开发提供了支持。UNIX 提倡开发小而精的程序，用户可以将小程序组装成一个大的应用系统。

随着 UNIX 的普及和发展，UNIX 正在成为操作系统的标准。

1.1.1　UNIX 的起源及发展

UNIX 的起源可以追溯到 1965 年。当时，贝尔实验室、通用电气公司和麻省理工学院合作开发一个称为 Multics 的新操作系统。要求这个操作系统能支持多用户，提供支持多个用户同时进行数据处理的机制。到 1969 年，Multics 操作系统开始在 GE 645 计算机系统上运行，但许多最初的目标并未 达到。由于某种原因，贝尔实验室从该项目中撤出。贝尔实验室中许多参加过 Multics 项目的技术人员后来也参与了 UNIX 的开发。

UNIX 的产生首先要归功于 Ken Thompson 和 Dennis Ritchie，他们都是在贝尔实验室工作的科学家，也都曾参加过 Multics 项目。Thompson、Ritchie 及许多其它人（Rudd Canaday、JoeOssanna 和 Doug McIlroy）最先设计并勾勒了一个简明的系统，这就是后来第一个 UNIX 文件系统的最早框架。

Ken Thompson 编写了一些程序模拟他所提出的文件系统的功能，模拟 在请求调页环境下程序运行的行为，并为 GE 645 硬件编写了一个简单的操 作系统内核。他还编写过一个称为“空间旅行”的游戏程序。该程序是在 Honeywell 635 计算机的 GECOS 操作系统下用 FORTRAN 语言编写的。在游戏程序的调试过程中，他发现该程序运行开销很大，而且对在程序中控制“宇宙飞船”的困难有了切身的体会。

Thompson 发现了一个不被人注意，也很少有人使用的 DEC PDP－7 计算机，很快决定把它作为空间旅行游戏程序的开发系统。这个机器机时很便宜，而且有较好的图形显示功能。游戏的开发使 Thompson 很快熟悉了机器硬件 的内部细节。由于没有可用的编译器，程序要先在 Honeywell 的 GECOS 系统上交叉汇编，将结果传输到纸带上，再由 PDP－7 读入，因此，开发工作很不方便。这种不利状况也促使了 Thompson 动手编写自己的系统。

Thompson 和 Ritchie 改进了他们的设计，并开始在 PDP－7 计算机上实现。这些设计包括 UNIX 文件系统的第一个版本，一组实用程序和一个进程控制 子系统。不久，系统开始运转，由于 PDP－7 系统已经可以支持自身，因而也就不再需要利用 GECOS 作为开发支持系统了。

Thompson 他们开发的这个系统与 Multics 相比可以说是独具特色。Multics 开发人员多，功能庞杂，追求大而全；而 UNIX 开发人员少，功能清晰，追求小而精。因此，小组的另外

一个成员 Brian Kernighan，将这个新的操作系统戏称为 UNIX，这是相对于 Multics 的双关语。(在英语中，uni－这个前缀表示“一个”、“单独”的意思，而 multi－则表示“多个”的意思；而 UNIX 的 X 与 Multics 的 cs 发音又相同。)

UNIX 操作系统的第一次实用是为满足贝尔实验室专利部对正文处理的要求，时间是 1971 年，运行机器是 PDP－11。第一个版本的 UNIX 系统是用汇编语言编写，只要求 16K 的操作系统空间，8K 的用户程序空间，一个 512K 的磁盘驱动器。在系统中，每一个文件长度不能超过 64K。

一开始，Thompson 准备为 UNIX 实现一个 FORTRAN 编译器，但随后又转去搞一个称为 B 的新语言。B 语言的最大缺陷在于它是解释执行的，运行时效率较低。Dennis Ritchie 对解释执行的 B 语言作了改进，提出了编译的 C 语言。C 语言中允许定义数据结构、说明数据类型，表达能力很强。同时，C 语言又保留了一些较低级的语言特性，很容易编译成效率较高的运行代码。因此，C 语言非常适于书写系统程序。1973 年，Ritchie 将整个操作系统用 C 重写了一遍，这就使 UNIX 成为第一个从汇编语言表达转到用高级语言书写的操作系统。这样，只要提供了相应的 C 编译器，就可以很方便地将 UNIX 移植到任何其它类型的机器上。

安装 UNIX 的机器越来越多，开始仅局限于贝尔实验室内部。由于内部支持，贝尔实验室还专门成立了 UNIX 系统组。当时，根据 AT&T 与联邦政府 1956 年签署的协议，AT&T 还不能合法地经营计算机产品。AT&T 并未作任何广告，也未试图向市场推广 UNIX，AT&T 只是向各大学和一些商学院赠送了一些 UNIX 系统，包括源代码。UNIX 简明清晰的特性使它很受用户，特别是程序员的喜爱。在以后的几年中，UNIX 系统的装机数量稳步增长，其影响也与日俱增。

1977 年，UNIX 被首次移植到非 PDP 的硬件：Interdata 8/32 机上。移植只须作很少的改动，用 C 改写 UNIX 的好处得到了验证。很快，另外一些公司也开始把 UNIX 移植到其它机器上，并对 UNIX 作了许多改进，使它更适合于用户的需要。结果，从基本系统演化出许多变种来，即使在 AT&T 内部也有几种变种，为统一 UNIX 的版本，贝尔实验室把它们结合到一个标准的系统中，这就是通常所说的 UNIX 系统Ⅲ。

目前所能看到的 UNIX 最早版本是 UNIX 版本 6，其次是版本 7，接下来是系统Ⅲ。UNIX 系统 V 是从未发布的内部版本系统Ⅳ演化而来的，由贝尔实验室于 1983 年 1 月发布。

UNIX 的一个主要变种是 BSD 系列，主要版本是 BSD 4.3，是由加州大学伯克利分校计算机系开发的。BSD 对 UNIX 作了许多改进，提供了独立于具体终端类型的正文编辑器 vi，增强了 UNIX 对网络的支持，首先在 UNIX 中对 DARPA 的 TCP/IP 网络协议提供支持，为 UNIX 的存储管理引入了请求调页的策略，极大地改进了 UNIX 的效率。

UNIX 在微机领域的主要变种是 XENIX 系统，由 SCO 公司开发。目前，XENIX 已发布到版本 3.2. XENIX 提供了与 UNIX 基本相同的功能，稍作了一些裁剪。照目前的趋势看，XENIX 最终将与 UNIX 系统 V 合为一体。

目前，UNIX 系统 V 已发布到第四级，而且有众多公司的各种移植版本。这些公司中有 AT&T，Microsoft，IBM，Amdahl，Hewlett－Packard，Sun Microsystems，DEC，Microport，Bell Technologies 和其它一些公司。UNIX 系统 V 4.0 版已于 1989 年 10 月发布，而且提供了新的图形用户界面。该界面称为 Open Look 由 AT&T 和 Sun Microsystems 联合开发。

UNIX 的其它重要变种有 OSF 的 OSF/1，Concurrent Computing 的实时 UNIX RTU，以

及卡内基－梅隆大学的 Mach。这些系统的主要研制目的是使 UNIX 支持实时应用，支持并行/分布的体系结构。

本书所讨论的 UNIX 系统 V 2.0 是由 AT&T 公司研制的，主要运行于 AT&T 的 3B 系列计算 机上。3B 系列计算机于 1984 年推出，在美国颇有影响。

1.1.2 UNIX 的特点

UNIX 既是一个操作系统，又是一个程序设计平台。UNIX 在物理计算机上，构造了分层的虚拟机，为用户提 供了独立于具体机器硬件的各种服务。UNIX 所提供的各层虚拟机可用下图表示：

应用层
用户/Shell 层
内核层 Kernel
硬件层

UNIX 分层体系结构

UNIX 的最大特点是它的核心短小精悍。UNIX 的设计者认为，操作系统的核心应当只完成最基本的功能，应该是提供机制而不是提供具体的服务。实际上，UNIX 提供的许多功能都是由进程，而不是由核心来提供的。

UNIX 的另一个主要特点是它使用户以一种统一的观点看待系统的资源和提供的服务。在 UNIX 用户看来，UNIX 的文件和设备都是一种无结构的字符流。都可以用同样的方式打开，关闭，读，写。这种统一的观点也贯彻到对进程的看法中。用户输入的各种命令，都通过系统启动的进程进行相应的处理。系统提供的许多服务也完全是由特殊的进程来处理的。

UNIX 还提供了文件系统的装卸功能，使用户可以同时使用多个文件系统，并且，可以很方便地对这些文件系统加以组织。同时在具体使用时，又感觉不到多个文件系统的存在。极大地提高了文件系统的灵活性和安全性。

UNIX 提供了丰富的进程间通讯手段。UNIX 的进程总是从标准输入读，向标准输出写，但它并不关心具体是从哪儿读，向哪儿写。这样，用户就可以利用管道，倒向等手段进行相关的处理。目前的 UNIX 提供了软中断信号，信号量，消息队列，共享内存等多种进程间通讯手段。

UNIX 的简洁、统一的思想也反映在 UNIX 对程序设计的支持上。UNIX 鼓励程序员开发只完成一种功能的程序。因为，这种程序易于编写和调试，效率也容易保证。程序员可以利用 UNIX 提供的机制将这些小的程序组装起来，共同完成一件比较复杂的任务。

UNIX 的另一个特点就是它是用高级语言书写，并且在开始的时候，又是以源代码的形式发布的。以高级语言书写，使它易于移植；以源代码的形式发布，使所有关心它的人都可以自己动手在系统上增加新的功能。特别地，UNIX 在高校首先开始流行，许多学校以此作为学习操作系统的参考，这就产生了所谓的四年效应，客观上促成了 UNIX 的流行。

UNIX 是一个开放的操作系统，它允许用户自己编写工具，并以一种非常自然的方式加入到系统中。UNIX 所提倡的开放性正在形成潮流，而且其影响已经波及到整个计算机界的各个领域。

1.2 UNIX 核心概述

本节首先介绍 UNIX 核心的体系结构，主要从系统构成及用户使用的角度来进行考虑。然后介绍核心中各个子系统的功能。

1.2.1 核心的体系结构

1.2.1.1 从系统构成角度考虑

整个 UNIX 系统从系统构成角度看可大致分 3 个部分：用户级，核心级，硬件级。用户级包括用户编制的应用程序，UNIX 系统提供的实用程序（各种 shell，编辑器，各类语言的编译器等），标准的函数库等。系统调用是一般用户取得核心服务的唯一手段。用户在调用系统调用时实际上是通过标准库函数来完成。标准库函数中有每一个系统调用对应的函数。这些函数通过特殊的系统陷入指令（在我们分析的系统中为 GATE 指令）进入到核心态请求核心的服务。它们起的主要作用是传递系统调用的参数以及将核心的执行结果返回给用户程序，系统调用的详细过程请参见第 7,13 章。

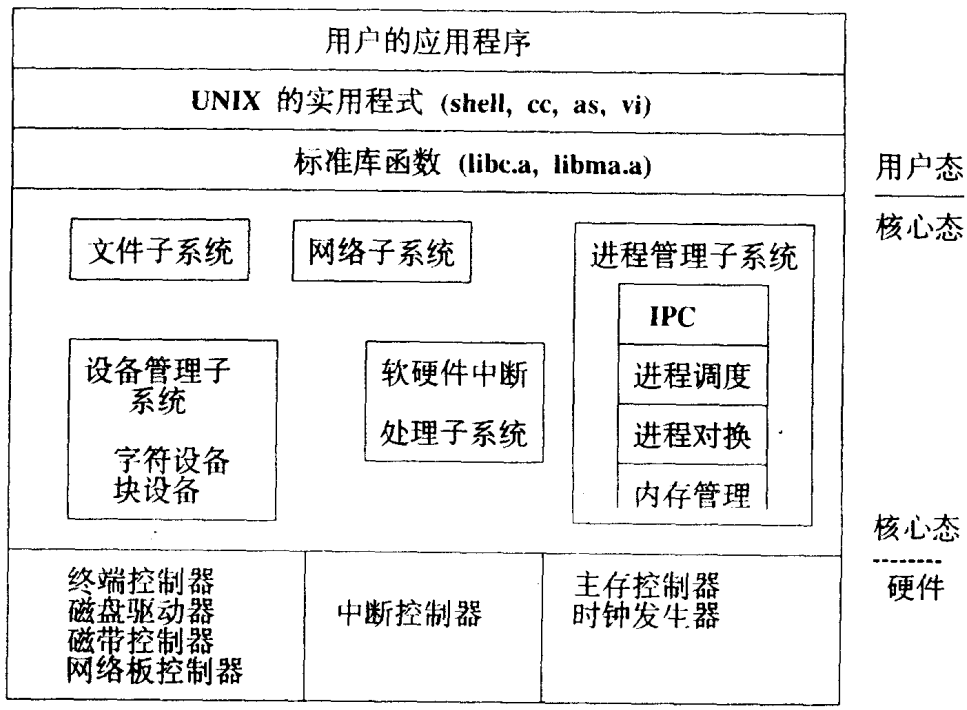

图 1—1

UNIX 的核心是联系用户级与机器硬件的桥梁。其主要的工作是完成进程及文件系统的管理工作。核心从功能上可划分为 7 个主要的模块：文件子系统，进程调度与对换，进程之间的通信（IPC），内外存管理，设备管理，软硬件中断处理以及网络子系统。这几个模块是从逻辑上进行划分的，核心实现时由于它们之间互相交叉调用，模块的划分非常模糊。每一个模块的功能将在 1。2.2 节分别给予介绍。

硬件级包括了整个裸机。即 CPU，总线，时钟，主存，外存（磁盘，磁带），各种 I/O 设备的控制器等。在第 2 章我们较详细介绍了 AT&T 3B5 机器的硬件情况。

1.2.1.2 **从数据结构角度考虑**

图 1－1 从核心构成角度介绍了核心的组成，图 1－2 将从核心的数据结构角度来进行描述。

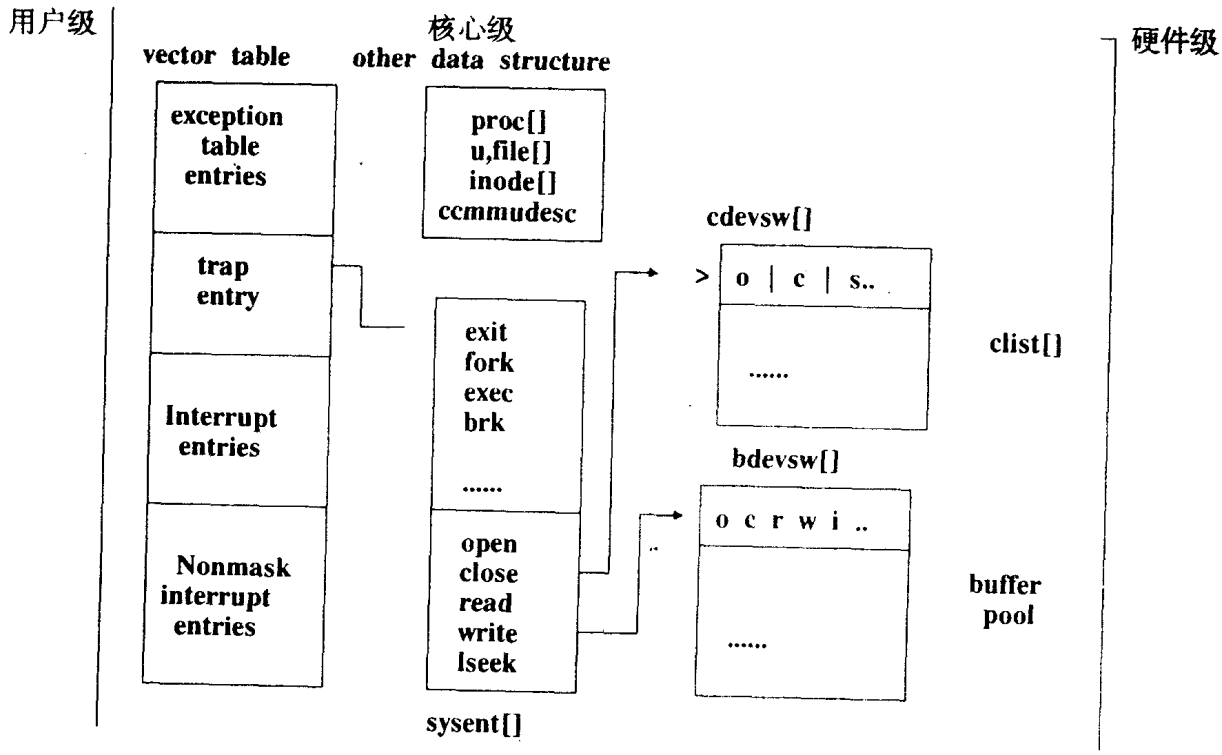

图 1－2

从核心外进入到核心只能通过各类中断向量表格(vector table)进入。" execptiontable entries" 是例外处理的入口地址表，例如 CPU 复位例外请求，栈溢出例外等。"trap entry" 是系统调用陷入的统一入口地址表。"Interrupt entries" 是一般可屏蔽中断的入口地址表(包括各类设备的中断处理程序的入口地址)。"Nonmask interrupt entries" 是不可屏蔽中断的入口地址表（如：掉电，总线错误等）。"sysent[ ]" 是各个系统调用的入口地址表。"cdevsw[ ]" 是字符设备的驱动程序的入口表。"bdevsw[ ]" 是块设备的驱动程序入口表。"buffer pool" 是用于进行块设备 I/O 的缓冲池。"clist[ ]" 是用于字符设备(tty) I/O 的缓冲链表。核心中除上述的数据结构外，其他的数据结构可参考第 13 章中的图示。

系统调用的过程实际是通过 "trap entry" 中存放的"sysent[ ]" 系统调用统一入口地址并且根据系统调用的参数在 sysent[ ]表中找到相应程序的地址并执行的过程。当系统调用与 I/O 操作相关时，将根据打开文件是字符文件或者块文件分别通过 cdevsw[ ], bdevsw[ ] 表中存放的设备驱动程序进行 I/O 操作。字符设备驱动程序将用到数据结构 clist[ ]，块设备将用到 buffer pool。

1.2.2 核心各子系统概述

1.2.2.1 文件子系统

UNIX 文件子系统管理一般文件，目录文件及设备文件。从用户角度看，一般文件是一个无格式的字符流文件，用户无需了解核心是按什么数据结构来管理文件的。用户可以按他自己的意愿去解释字符流，这种解释与操作系统如何存储数据无关。目录文件记录一般文件、特殊设备文件并将整个文件系统组织成一个树状的结构。设备文件代表每一个 I/O 设备，用户可以按一般的文件对待它。

用户只能通过与文件相关的系统调用对文件系统中的各种文件进行操作。主要的系统调用是：open(打开一个文件)，creat(建立一个新文件)，mknod(建立一个设备文件)，close(关闭一个打开的文件)，chown(改变文件的所有者权限及组号)，lseek(当前文件的读写头指针移动)，write(将数据写入当前文件)等。

从文件系统的核心构造来看，主要由 3 类数据：用户文件描述字表(在 u 区定义)，系统文件表(在核心虚空间定义)，索引节点表(内存中的节点表在核心虚空间，每一个文件系统在磁盘上也有索引结点表)组成。对文件的操作实际上是针对它对应的索引结点的操作，不同的文件类型，对它的操作也不相同。文件子系统的详细描述请参见第 3,4 章。

1.2.2.2 进程子系统

对用户来讲，进程就是一个程序的执行，一般用户在 shell 状态下键入一个可执行文件名时，该文件即以一个进程的形式在 UNIX 系统的管理下执行，完成用户的任务。从 UNIX 核心来看，进程是一个占有一定系统资源(CPU，内存空间，外存空间等)的独立的运行实体。一个进程在其生存的整个周期可能要经历多种状态：运行态(当前进程正占据 CPU 运行)，就续态(进程处于可运行状态，但由于已有其他进程占据 CPU 在运行，它本身处于等待 CPU 状态)，睡眠态(进程由于申请系统资源而未能满足的情况下进入睡眠状态，一旦申请的资源被满足，它将进入到就续态)，僵死态(进程释放了它占据的大部分系统资源，只留下它使用系统资源的记录，该进程的父进程或者系统进程 1 将记录它的使用系统的信息并且最终释放它的所有资源)。在第 5 章中将详细介绍所有的状态以及它们之间的相互转换。

核心可通过下述各种操作完成进程从创立到终止的一系列工作。

fork：创立一个新的进程。

exit：释放进程占据的系统资源，进入到僵死态。

kill：向进程发软中断信号，告诉进程外界发生的事件。

signal：告知进程处理软中断信号的方式。

wait：进程处于等待状态直到接收到特定的软中断信号。

sched：进行两个进程之间的 CPU 切换。

nice：改变进程运行的优先权，使进程申请系统资源的能力提高或下降。

pause：暂停一个进程的执行。

一个进程的主要数据结构是核心给它分配的进程登记表(u 及 proc[ ] 结构)。实际上进程的登记表很好地反映了进程本身的定义。在进程登记表中大约有如下的内容(完整的描述在第 5 章中介绍)：

- 进程的名称；
- 处理器状态(程序计数器，CPU 寄存器内容，中断优先级等)；
- 当前进程的状态(运行态，就续态等)；

- 进程的优先权（申请系统资源：CPU，Memory，IO 等）；
- 存取系统资源的权限；
- 虚拟存储管理信息；
- 记账信息；
- 其他信息；

进程的主要部分将在第 5 章中介绍，第 6 章将介绍进程的软中断信号的处理，第 10 章介绍进程之间的通信（IPC）。

1.2.2.3 设备子系统

设备子系统完成进程与外部设备如磁盘，磁带，终端，打印机及网络进行通信的任务。从用户角度看，设备与一般的文件没有多大的差别。用户对设备的存取通过文件系 统的有关系统调用来进行。在核心中，设备子系统又被称为设备驱动程序。通常的设备驱动程序与设备类型是一一对应的：一个磁盘驱动程序控制系统中所有的磁盘，一个终端驱动程序控制所有的终端，一个磁带驱动程序控制所有的磁带。除与物理设备相关联的驱动程序外，UNIX 系统还支持＂软设备＂的概念。例如在第 5 章中介绍的核心直方图统计就是通过一个软设备驱动程序来完成。

核心中驱动程序的界面是由块设备开关表（bdevsw[ ]）与字符设备开关表（cdevsw[ ]）进行描述的。每一种设备类型在表种有若干表项，这些表项在系统调用时引导核心转向适当的驱动程序入口。打开，读，写块设备将导制调用设备在 bdevsw[ ] 表中对应的相应程序。字符设备的 read，write，ioctl 将调用 cdevsw[ ] 表中的相应表项。

硬件与驱动程序的接口是由与机器有关的控制寄存器或操作设备的 I/O 指令以及中断向量等组成。当一个设备中断出现时，系统识别发出中断的设备并调用 适当的中断处理程序。软设备的中断处理程序一般在时钟中断处理程序中执行。

核心中块设备驱动程序之上的是高速缓冲池数据结构，它的主要作用是解决存取外部块设备与存取内存两者之间速度不匹配，外部设备慢的问题，同时可减少 存取外设备的次数。在字符设备之上的是字符缓冲链结构 clist[ ]，它与高速缓冲池具有相同的效用。

在本书第 8，9 两章，我们将分别介绍字符设备与块设备的驱动程序以及核心中与它们相关的数据结构。

1.2.2.4 存储管理子系统

存储管理子系统对一般用户来讲是看不到的，它的主要功能是对内存的空闲空间，外部设备对换区空间（一般为磁盘 ）以及虚拟存储空间进行管理。

内存的空闲空间是指机器的物理存储器中还未用到的空间。一般机器是在磁盘上开辟一个分区称为交换区用于存放内存中暂时不运行的进程的映象，磁盘上的交换区也可以看成是内存空间的扩展。内存以及交换区空间的管理是通过两个数据结构 coremap[ ]，swapmap[ ]以及围绕在它们周围的子程序所组成。

物理内存空间是很有限的，UNIX 操作系统为在有限的物理空间情况下提高机器接收的进程个数，提高 CPU 的使用率，减少对单个进程大小的限制以及在各进程空间之间提供保护（支持多用户，多道程序的设计），使用虚拟存储管理技术。根据进程映象与交换区的对换策略的不同又可划分为基于页式和基于段式的虚存管理系统。前者将对换的单位固定（页长），进程的映象存储在内存及交换区中不连续的空间中，当进程执行到某一个页 所涉及到的地址而该页不在内存时，将发生缺页请求中 断，对应的中断处理程序将把需要的

页调到内存。当内存空间不够时，UNIX 的核心将把某些进程的页对换到交换 区中。段式存储管理是将整个进程的映象在交换区与内存之间相互对换。我们分析的系统属于这种类型，它详细的实现将在第 6 章中介绍。

1.2.2.5 **软硬中断子系统**

我们将在第 7 章中详细介绍软硬中断子系统。该子系统进一步可划分为四个方面：硬件中断处理，软件中断处理，例外处理以及系统调用的实现。

硬中断是由于 CPU 之外的其他设备引发请求 CPU 服务的信号。当外部设备引发的中断请求被接受时，中断的处理实际为暂停 CPU 当前进行的工作，转到由中断向量表指向的中断处理程序去执行，处理完中断后再返回到 CPU 被中断的现场继续执行。

软中断对应的英文为 signal，在我们分析的系统中还有一类软中断我们称它为可编程中断 (software interrupt)。前者被用于进程之间互相发送信息，当一个进程接受到一个软中断信号时并不马上进行处理，而是在从核心态转换到用 户态的过程中进行处理。一些软中断信号用户自己是不可控制其处理的，核心要按事先规定的缺省处理，有些软中断信号的处理可以由用户自己控制。可编程中断与硬中断相类似，通过执行特殊的机器指令或者给某一个设备发送一个信息将产生与硬中断相同的信号。可编程中断与硬中断唯一的不同在于前者由外部设备产生，后者由 CPU 自身的执行所引起。

例外是指当 CPU 执行过程中发生栈溢出，地址出错等错误情况。例外处理不可屏蔽，其处理与中断类似。

系统调用（有些书翻译为陷入 —— trap）是用户程序请求核心服务的手段。不同的 CPU 有不同的 trap 指令，在我们分析的系统中为 GATE。该命令的主要功能是使用户的程序进入到核心虚空间并执行核心的程序。第 7 章将详细介绍整个系统调用的过程。

1.2.2.6 **进程间通信子系统**

进程间通信子系统（Interprocess Communication）允许任意的进程之间交换数据和同步地执行。在文件子系统 中将介绍用有名及无名管道进行进程间通信的过程。在软硬中断子系统中将介绍用软中断(signal)进行进程间通信的过程。在 AT&T 的 UNIX SYSTEM V 中使用三种机制：信息机制（message），共享存储区机制（shared memory）以及信号灯机制（semaphore）。在由 Berkeley 开发的 UNIX 版本 BSD 中还提供了 SOCKET 机制用于进程之间的通信。

信息机制允许进程发送格式化的数据流到任意的进程。共享存储区机制允许多个进程之间共享他们虚地址空间中的 部分或者全部的数据区域。信号灯机制用于多个进程之间的同步执行。这三种通信机制将在第 10 章中被详述。

1.2.2.7 **网络子系统**

网络子系统介绍支持网络通信的在核心中的低层设备驱动程序。对于虚电路或报文转发协议的支持是通过系统的库函数来完成的。

在 3B 系列的计算机中，都使用了一种称为 NI 的设备(NI 即 Network Interface)。NI 设备是一块电路板，利用硬件对以太网协议及 IEEE 的 802.2/802.3 协议提供了支持。

系统共提供了两组设备驱动程序，一组是 NI 设备驱动程序，一组是 3b 设备驱动程序，其中，3b 设备驱动程序是支持虚设备，它利用 NI 设备驱动程序提供一组更高级的服务。有关网络子系统的其他信息请参照第 11 章。

1.3 小结

本章对 UNIX 从整体及核心的构成上进行了描述，可以作为阅读后面几章的基础。第二章介绍我们分析的 UNIX SYSTEM V 2.0 核心所运行的机器 AT&T 3B5 的 CPU 构成特点，机器指令以及 C 语言及汇编语言函数调用的有关规定。第三章介绍文件系统相关的数据结构以及较低层子程序的算法。第四章介绍与文件系统相关的系统调用以及核心中相关子程序的实现。第五章介绍进程的概念，与进程相关的数据结构以及进程控制程序的实现算法。第六章介绍虚拟存储管理的概念，策略，算法及共享正文段，双映射段的实现算法。第七章介绍中断，例外，系统调用(陷入)，软中断的处理过程，对时钟中断处理程序以及软中断处理涉及的算法进行了分析。第八章介绍字符设备驱动程序。第九章介绍块设备(磁盘)驱动程序。第十章介绍进程间通信机制：消息，共享存储区及信号量。第十一章介绍网络通信机制。第十二章介绍核心的装入及核心初启的过程。第十三章总结核心中的主要数据结构并且通过一个实例来跟踪核心的整个运行过程。附录 A 给出 UNIX 操作系统上可执行文件的通用格式 COFF。

第二章　WE32100 微处理器硬件介绍

2.1　概述

本章将介绍 WE32100 处理器与操作系统设计相关的部分。一般编程的方法请参考 AT&T 3B5 机器的随机手册。我们下面介绍的内容是 CPU 可支持 的各类功能或者说操作系统设计者可使用的各种功能，在我们分析的操作系统 AT&T 3B5 UNIX SYSTEM V release 2.0中有些在本章中介绍的硬件功能并没有被用到。

2.1.1　CPU 的组成

WE32100 微处理器是寻址与数据总线互相分离的 32 位微处理器。该总线可以在物理或者虚拟空间状态下对 4 千兆字节(2^{32}) 的主存或者设备地址进行存取。数据的读写是通过 32 位的双向数据总线来完成，传送的数据可以是 8 位，16 位 或 32 位长，总线内部按统一的 32 位格式来处理。

处理机的执行速度由执行的指令序列及 CPU 内部存放指令的 cache 所决定。CPU 采用指令的流水线互迭执行技术使其效率进一步得到了提高。当一条指令在执行过程 中发生中断时，CPU 可方便地找到被中断的指令并且重新该 指令的执行。这一特点可很好地支持缺页中断请求的虚拟存储管理技术。

WE32100 处理器主要由如下四个部件组成：主控器，cache 及其控制器，执行部件以及总线部件的控制器。主控器负责取指令及解码并且通过特殊的微指令序列控制 cache 及执行部件的动作。主控器还负责响应及控制中断及例外的处理。

图 2－1 如示 WE32100 芯片的构成。

cache 部件处理指令流并完成操作数的存取工作。它由控制器，指令 cache ，指令队列，立即与偏移量寻址的部件以及一个地址计算部件构成 (AAU) 。cache 控制器负责控制 cache 内部的各部件。指令 cache 大小为 64 * 32，与 CPU 的其他部件被设计在一个芯片中。其主要功能：减少 CPU 对内存的访问次数从而提高 CPU 的执行效率。当需要从内存中取指令时，指令数据将被放到指令 cache 及指令队列中，当刚执行的指令数据再一次需要，CPU 将从 cache 而非内存中访问它。指令队列 是一个 8 字节的先进先出队列，它存放当前 CPU 要执行的指令。当该队列为空时，cache 控制器将自动将它充满。立即与偏移量寻址部件给 AAU 提供计算 32 位地址所需要的数据。

CPU 的执行部件完成所有的算术逻辑运算，位移及循环位移操作以及测试不同的条件标志位。它包括：

- 执行控制器：控制内部各部件的执行。
- 16 个 32 位用户可访问的寄存器，9 个通用寄存器 (r0 － r8)，7 个特殊用途的寄存器 (r9 － r15)。
- 不对用户开放的 CPU 内存使用的寄存器。
- 32 位算术逻辑运算部件。

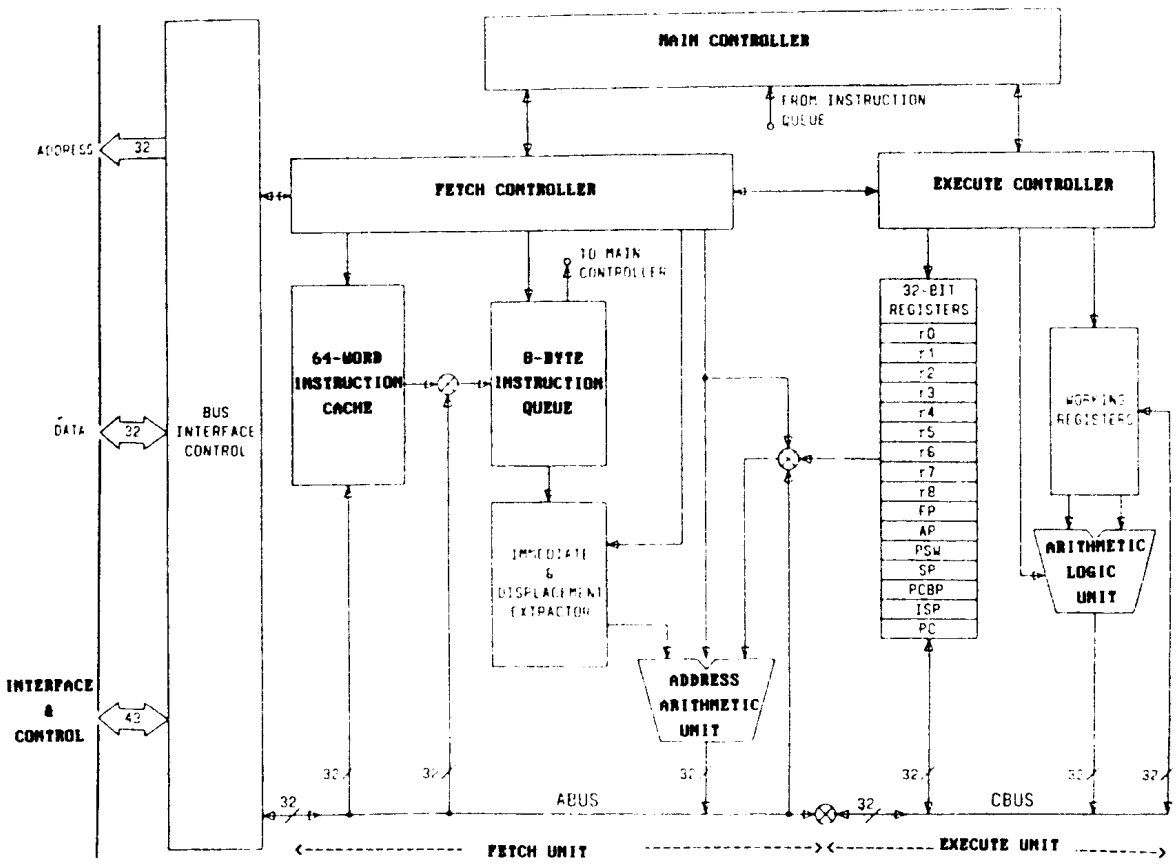

图 2－1 WE32100 CPU 构成图

2.1.2 用户可使用的寄存器

图 2－2 给出 16 个 32 位的寄存器 (r0 － r15) 的程序设计模型。这些寄存器是为高级语言的快速运行而设计的。除程序计数器(r15)，状态字 (r11)，进程控制块指针 (r13)，中断栈指针 (r14) 外，其他的寄存器可以在任何状态 (核心态或用户态) 被用户使用。上述几个寄存器可在任何状态下被读，要写的话，必须在核心态。

9 个通用的寄存器(r0－r8)可被用来进行高速的累加，地址的寻址及作为数据的临时存放地。前 3 个寄存器 (r0－r2) 在 C 编译程序中被用来存放表达式计算的中间值，它们在子程序调用中还完成参数及返回值的传递工作。例如 r0 永远返回子程序的返回值，r0 和 r1 两个寄存器返回浮点值。如果一个程序返回一个结构的指针，则返回值由 r2 来指定。

桢指针 (FP － r9) 指向运行栈中一个函数内部变量的开始位置。该指针隐含地被 SAVE 及 RESTORE 机器指令所改变。

参数指针 (AP － r10) 指向当前运行栈中一个函数的实参的开始地址处。该指针隐含地被指令 CALL 及 RET 所改变。