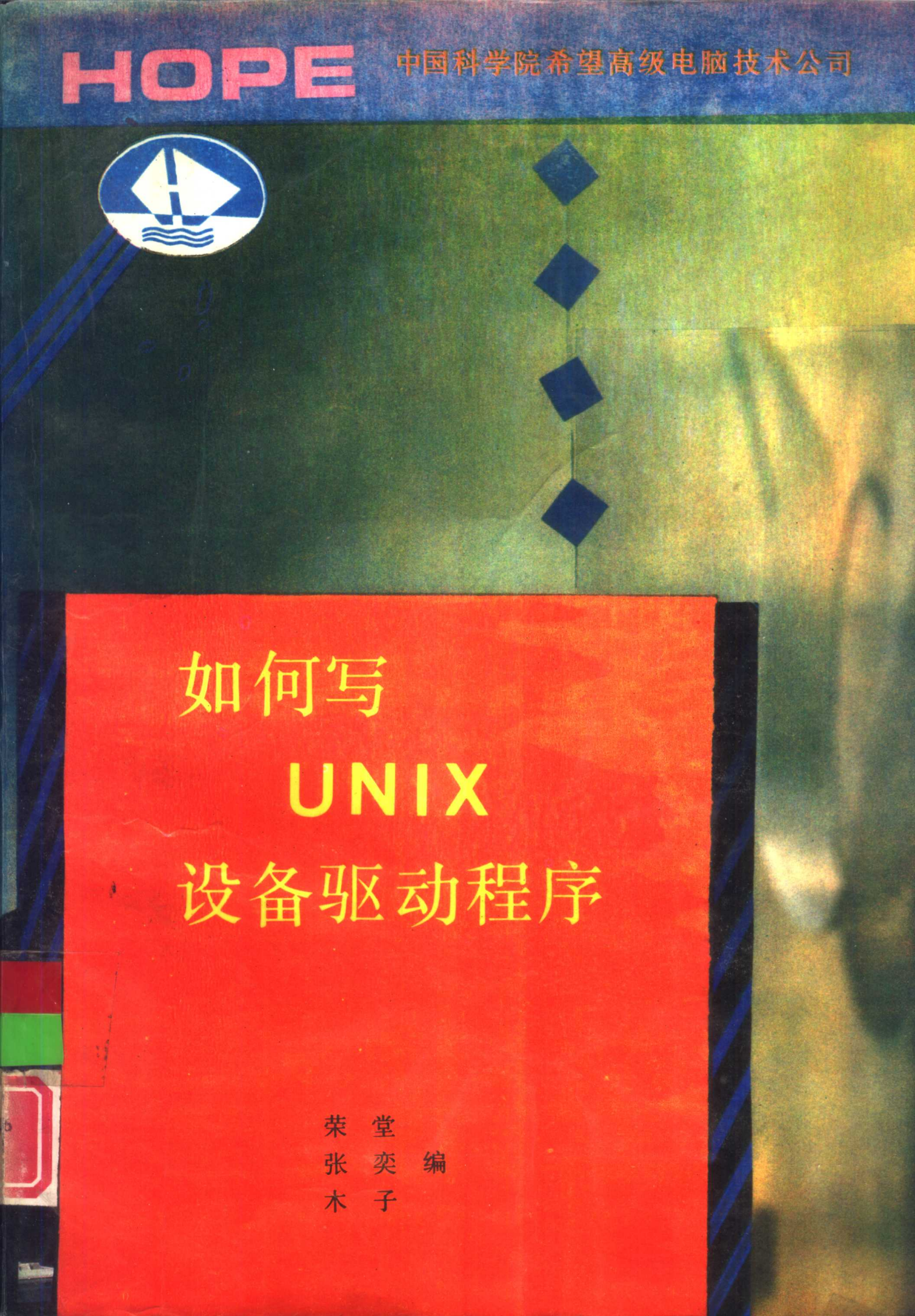
HOPE 中国科学院希望高级电脑技术公司



如何写 UNIX 设备驱动程序

荣 堂
张 奕 编
木 子

如何写UNIX设备驱动程序

荣堂　张奕　木子　编

中国科学院希望高级电脑技术公司

一九九一年五月

■ 北京市新闻出版局
准印证号：3320—90320
■ 订购单位：北京8721信箱资料部
■ 邮　　码：100080
■ 电　　话：2562329
■ 传　　真：01—2561057
■ 乘　　车：320、332、302路
车至海淀黄庄下车
■ 办公地点：希望公司大楼一楼
往里走101房间

前　言

UNIX操作系统是当今最流行的操作系统之一，几乎所有的计算机系统，从微型机到工作站，小型机，无不把UNIX作为基本的操作系统而运行。随着UNIX的广泛使用及标准化，在UNIX环境下开发各种新型的标准计算机板级产品外部设备，已经成为当前计算机系统开发中的一个热门课题，而这里涉及到的一个基本问题，就是如何写UNIX设备驱动程序的问题。

本书是第一本全面、系统地介绍如何在UNIX 环境下写设备驱动程序的专著，它从MASSCOMP RTU（MC 68×××系列机器上实现的UNIX）为背景，对写UNIX设备驱动程序所涉及到的基本内容作了详细、具体的介绍，并给出了大量的程序例子，是一本难得的参考书和工具书。

尽管国内也有介绍UNIX设备驱动程序的资料，但基本都是零星、分散的，全面介绍UNIX设备驱动程序的中文专著还没见过。据此，译者将这本书翻译出来，介绍给国内从事计算机和UNIX开发及研究的同行，以弥补国内这方面中文资料缺乏的不足。希望它能对国内计算机及其外设、板级产品的开发有所帮助，限于译者水平，书中不妥之处，还望提出宝贵意见。

本书的出版，得到了希望公司秦人华经理的大力帮助。译者对此表示深深的谢意！

译者

1991年1月31日

目　录

第一章 UNIX及其I/O子系统

1.1 UNIX操作系统 …… (1)
1.2 从用户角度看UNIX …… (5)
1.3 从程序员角度看UNIX …… (5)
1.4 进程控制和调度 …… (10)
1.5 系统调用 …… (12)

第二章 UNIX I/O系统

2.1 文件系统 …… (14)
2.2 关于文件操作的系统数据结构 …… (17)
2.3 块缓冲系统 …… (19)
2.4 设备驱动程序 …… (20)
2.5 通过系统的I/O请求流 …… (21)
2.6 驱动程序综述 …… (21)

第三章 I/O硬件和设备驱动程序

3.1 I/O体系结构 …… (28)
3.2 I/O设备的特征 …… (31)

第四章 系统生成

4.1 核心与驱动程序的接口文件 …… (34)
4.2 系统配置数据文件 …… (35)
4.3 名字构成规则 …… (38)
4.4 配置表文件—conf.c …… (39)
4.5 硬件接口文件 …… (44)
4.6 构造一个新的核心 …… (46)
4.7 创建设备特殊文件 …… (47)

第五章 运行时的数据结构

5.1 虚拟和物理地址 …… (48)
5.2 标准I/O数据结构 …… (49)
5.3 地址转换和数据访问 …… (56)
5.4 驱动程序与调用程序的相互作用 …… (64)
5.5 驱动程序内的同步化 …… (67)

第六章 驱动程序逻辑举例

6.1 设备的定义 …… (74)
6.2 设备数据结构 …… (76)
6.3 例1：同步字符输出 …… (77)

6.4 例2：表中的缓冲字 符 …… (79)
6.5 例3：系统空间缓冲器的 DMA输出 …… (83)
6.6 例4：用户空间的同步DMA …… (88)
6.7 同步I/O多路 …… (89)
第七章 驱动程序的开发方法
7.1 调试宏 …… (92)
7.2 跟踪驱动程序动作 …… (97)
第八章 样板字符驱动程序和样板块驱动程序
8.1 公共特性 …… (106)
8.2 样板字符驱动程序—chdriver …… (107)
8.3 样板块驱 动 程 序—bkdriver …… (111)
第九章 基本设备驱动程序要求
9.1 需要的入口点 …… (115)
9.2 入口点参数、动作和返回 …… (115)
第十章 专题
10.1 支持多设备 …… (122)
10.2 错误重发逻辑 …… (125)
10.3 磁带驱动程序 …… (127)
10.4 使用寄存器变量 …… (129)
10.5 编程注意事项 …… (129)
10.6 ASTs …… (129)
10.7 例子：采用AST的异步DMA …… (131)
附录A 执行头文件一览表 …… (138)
附录B 核心I/O 支持例程 …… (140)
附录C 样板字符驱动程序 …… (178)
附录D 样板驱动程序 …… (196)
附录E XENIX …… (210)
附录F 伯克利UNIX兼容性 …… (217)

第一章 UNIX及其I/O子系统

本章给出UNIX操作系统的综述及其为执行输入/输出（I/O）操作所使用的方法。已讨论过的某些课题，仅仅是为写驱动程序通读一下，并不需要知道其详细内容。其它课题是该科目的中心，但在最初的讨论中并没有详述。凡是过去任何地方作的模糊地描述，对于现今讨论的子题目来说，既不是驱动程序的中心，也不是以后章节中要详述的东西。

从二十世纪七十年代开始，UNIX操作系统已在贝尔(Bell)实验室（和以后的AT &T信息系统）和许多大学和公司里得到很大发展，贝尔实验室是最初开发的，而各大学和公司则将UNIX作为其基本的软件系统了。结果，任何两个由UNIX系统派生出来的系统，在其内部实现上有很大区别。尽管设备驱动程序的接口概念和UNIX的大多数版本相似，但UNIX操作系统的内部操作的完整过程超出本书讨论范围。

本书通过UNIX的MASSCOMP设备的RTU，提供编写驱动程序的内部操作的专用例子。我们还将引用其它常见的UNIX版本中的一些例子。

对于那些有兴趣研究如何使用UNIX系统的有关更多知识的读者来说，可在书目提要中查到有关UNIX的大量书籍。

在UNIX程序员手册的第2卷中，有：

· The UNIX Time Sharing System by D.M.Ritchie and K. Thompson

· UNIX Implementation by K. Thompson

The UNIX I/O System by D.M.Ritchie文章，从中可找到有关UNIX系统运行的更多的资料。

注意，这些文章是几年前写的，对现代UNIX未进行准确地极详细地描述，但提供了原开发者关于UNIX的概述。

1.1 UNIX操作系统

通常，UNIX软件系统由三部分构成：

1. UNIX核心
2. 数目不定的用户进程
3. 存储在第二存储器（一般是磁盘）上的数据文件系统。

UNIX核心是系统的心脏，仅仅是能直接访问和控制系统硬件的程序，硬件包括处理器，主存和I/O设备。用户进程响应终端上输入的命令并执行用户的任务。核心的功能是提供具有访问包括文件系统在内的系统资源的这些进程。为使整个硬件和软件系统起到多用户、多道处理的分时系统的作用，核心充当了协调程序的角色。

对从交互的终端上检验激活的UNIX系统的用户来说，其核心是透明的。全部的程序都能运行，包括编译程序,I/O实用程序以及终端命令解释程序,都是用户进程。对于编写应用程序的程序员来说，核心是以系统调用的方式呈现，这些和普通函数或子程序一样，但必须控制其执行动作及在系统范围基准上的协调。

用户进程具有执行共享系统资源需求的特定作业。一般来说，激活的多用户进程，存在

着资源竞争的问题。核心要维持其全局需求及所有用户进程的优先极，最后确定其服务顺序。

核心

UNIX核心是由唯一的大的可执行二进制映象组成。其中大部分模块由C语言编写，但也有一小部分是用汇编语言编写，核心的模块都是用标准UNIX程序进行编辑和汇编，然后链接成二进制内存映象。由引导程序将其装载到主存并启动执行。这儿对引导程序不作详细介绍。

核心有三个要完成的基本任务：

- 建立用户进程并调度其执行
- 提供系统服务
- 处理硬件中断和例外

以下是核心的逻辑综述：

- 核心维护已存在用户进程表中并与每个用户进程的优先数调度相连
- 检查进程表并将那些已准备好运行的进程从那些不准备运行的进程中分离出来。一个进程有可能因种种原因而不准备运行，多数情况下是因为要等待I/O操作的完成和等待其它进程结束运行，还可能是等待得到各种资源，如内存资源等。
- 如果这些都就绪，那么，具有最高优先极的进程授权执行。它得到中央处理器(CPU)的控制并暂时挂起核心的执行。

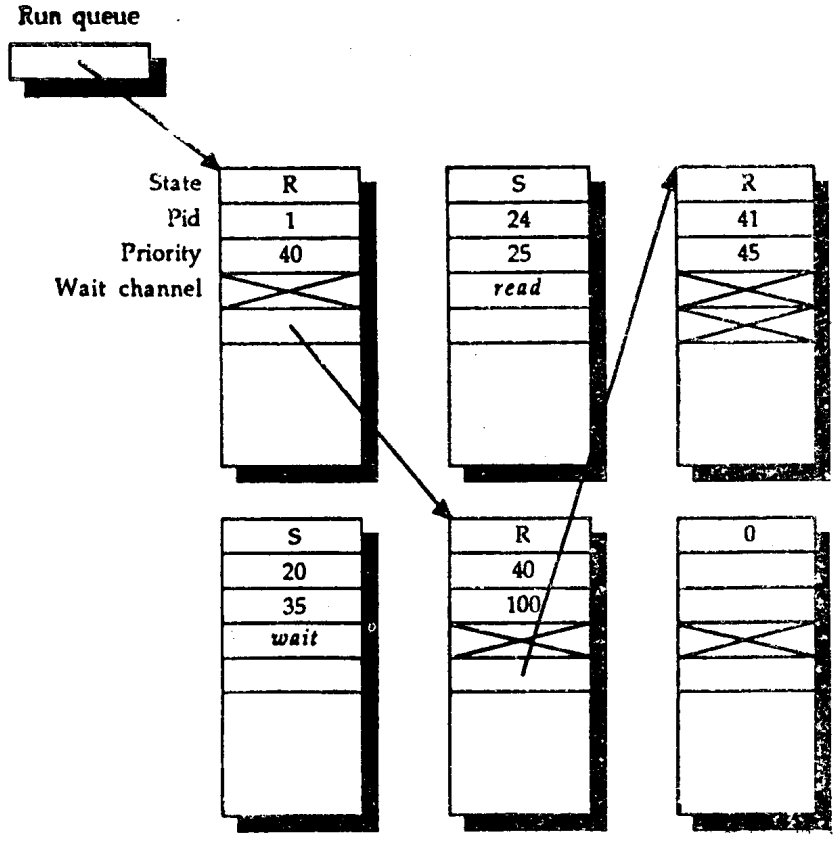


图1-1 进程表项

·被选中的进程一直运行到它产生系统调用；这时，该用户进程挂起，核心再次激活。核心处理系统调用，然后决定哪一个用户进程可以运行。

图1-1是一个简化进程表，进程1、40和41是可运行的（R），进程20和24在睡眠（S）等待特定事件，进程24等待读设备，进程20等待子进程的退出。

某些系统调用的请求可通过执行核心代码得到完全地满足。例如：lseek调用是设置落在这一级上的文件搜索指针。然而，对外围设备执行的I/O请求常常要对部分设备进行某些操作。在这种情况下，核心异步启动设备动作但并不等待其完成，请求服务的进程被标上未就绪标志，直到设备中断，标明I/O操作完成为止。

用户进程

在UNIX系统上运行的应用程序源代码将被编译和链接成二进制文件，该文件是已知的可执行文件。见UNIX程序员手册中的关于a.out(5)文件格式的描述。User Process是由UNIX核心执行的映象程序。

当一个映象由用户进程执行时，把可共享的指令集（正文段）装到内存的只读区。（核心将该区域置成只读，并不是硬件机制）把该进程的读/写数据区定在主存的二个附加段，第一个附加段放已命名的变量，第二个附加段作堆栈用。核心还要完成某些其它准备，然后通过把适当的值装到硬件程序计数器上来启动进程。

下面是有关进程在启动时刻状态的完整描述：

1. 进程在内存空间的内容包括代码区和数据区。
2. CPU通用寄存器程序计数器（硬件上下文）的内容。
3. 其它信息，如打开文件，当前工作目录，以及任何未处理的系统调用请求的状态（软件上下文）。

软件方面的信息被存储在由核心维持的数据结构中。

当一个进程被执行时，其代码和数据区是在主存里，而其寄存器和程序计数器值装在真正的CPU寄存器中。若必须挂起该进程，就把寄存器和程序计数器拷贝到内存的数据结构中，以便另一个进程能够使用CPU。

只要主存中保存着挂起的进程，就能通过把这些值装回CPU重新启动执行该进程。但是，核心也可能选择为其它用户腾空主存而把进程映象交换到盘上的举动。一旦这样作了，该进程，只有一小部分信息保存在主存中。依据这部分信息，核心足可以决定该进程何时重新执行和如何在盘上找到有关其它的信息。

实际上，一个用户进程是在假脱机上运行的，这种假脱机是由UNIX核心执行，并不为真正的系统硬件所识别。假脱机执行某些附加命令（系统调用），但并不允许访问机器的所有部分。例如，设备中断和特权指令等。如果一个进程被挂起，被交换到盘上，接着重新启动，这些事件对进程来说是完全看不见的，透明的，内存中属于核心和其它用户进程的代码区和数据区对用户进程来说也完全是看不见而透明的。

系统和用户地址空间

RTU是一个要求页式虚拟存储系统，每个用户进程有它自己的地址空间，从零开始至可能的大小（16兆字节或更大），这要看处理器的类型和系统提供的盘容量大小来确定地址的最大值。UNIX核心有它自己的逻辑特殊地址空间。这些地址空间分别称为用户和系统虚拟地址空间。在MC68000系统上，系统和用户的虚拟地址间是完全分离。用户虚拟地址空间

是系统虚拟地址空间的子集。

MASSCOMP硬件和操作系统将用户和系统虚拟地址空间映象到每页为4096字节的物理内存上。用户进程邻接的零基址的虚拟地址空间被映象到分散而不连接的物理内存页面上。因此，用户进程虚拟地址空间的个别页面可能由核心将其移到盘上，以便空出更多主存供另一个进程使用。见图1-2。

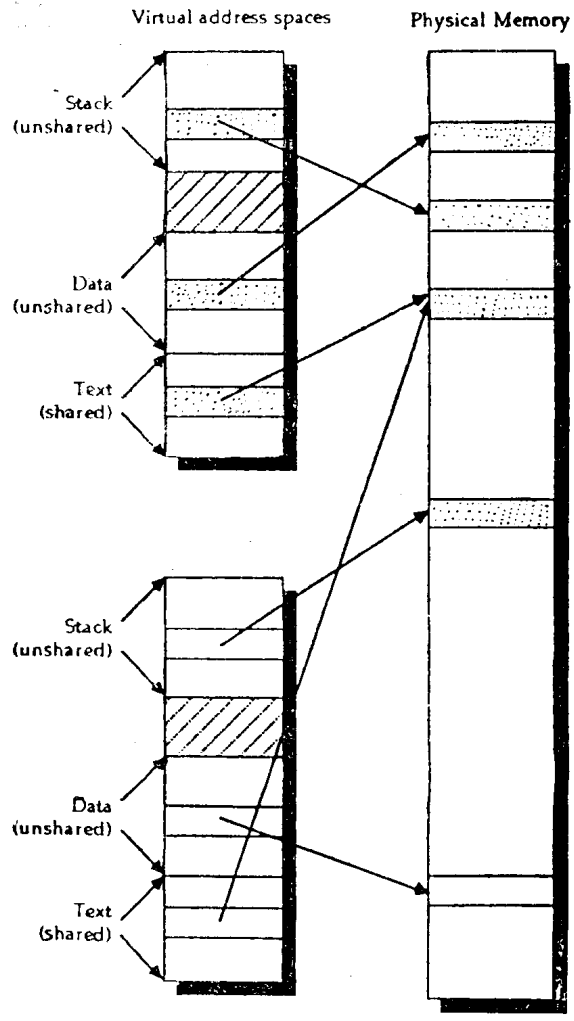


图1-2 虚拟到物理地址空间的映象

普通用户进程的程序员通常无需知道虚拟地址到物理地址映象的过程。因为核心和硬件是透明地处理这些事件。但，编写设备驱动程序的人，必须知道这些地址问题。驱动程序要同时安排用户进程地址，系统（核心）地址和第三种特殊类型的地址，I/O总线地址。本书详细具体地介绍了设备驱动程序所面临的各种情形之下出现的地址类型。同时，描述了核心函数，该函数是驱动程序为了将数据从一个地址空间拷贝到另一个地址空间以及为了产生在另一个地址空间没有拷贝的数据（这通过使用不同的地址映象表作的）调用的函数。

1.2 从用户角度看UNIX

UNIX系统最初设计和构造的是一个分时共享系统。因此，许多用户通过终端和UNIX系统进行交互对话。该终端是一个带有象打字机一样的键盘作为输入和一个用以显示字符的小电视屏幕或监视器作输出的设备。近期的UNIX版本已经增加了很多强有力的用户接口，诸如多窗口以及使用菜单或通过带有指示设备选择画面的调用操作的能力，但大多数主要的UNIX软件是通过终端进行。

用户发送具有数个参数的命令序列。命令参数（arguments）通常据约定，标识命令操纵的一个或多个文件（files），用连字符打头的参数是标记（flags）或选件（options）影响命令的动作情况，这些参数是作为正文串送到命令。

多数UNIX命令使用文件参数非常灵活：如果作为参数没有说明输入文件，则命令就从键盘上取输入，若未说明输出文件，则命令将输出写到终端屏幕上。例如，cat命令（concatenate的缩写）将任意个文件考贝到终端上命令：

```
cat preamble data
```

首先把preamble文件考贝到终端上，然后再把data文件考贝到终端上。

任何能够正常地从键盘输入和在终端上输出的命令都能通过命令解释程序重定输入或输出的方向。例如，ls命令列出一个目录的全部文件，命令：

```
ls -l
```

将在终端上列出当前目录下全部文件的长格式信息，象如下输出形式：

```
-rw-r-rw- 1 fred user 101  Jun 4 14:55 preamble
-rw-r-rw- 1 fred user 5428 Jun 4 14:13 data
```

这个信息可再次调用ls命令保留在文件中，把输出指向命名为dir的文件：

```
ls -l>dir
```

同样，wc是一个统计文件的字数、行数和字符数的命令。人们常常对wc说明一个文件参数，wc将参数作为输入，进行字数行数和字符数的统计。下面的命令说明，文件data重定向了wc命令的输入：

```
wc<data
```

最后，使用pipe（管道）可以把一个命令的输出用作另一个命令的输入。pipe是程序间传递数据的特殊机制。pipe对每一个程序来说，看上去就象是一个文件，命令行：

```
cat preamble data | wc
```

能对preamble和data文件中的字数、行数和字符数进行统计而不必建立一个临时文件。UNIX系统中的命令行常被称为管道线，因为命令行中含有由pipe连接的几个命令，管道线中的各命令（尤其是中间的命令）因为它们处理的数据是通过管道线流动进行的，所以叫作筛选程序（filters）见图1-3。

1.3 从程序员角度看UNIX

UNIX系统上的程序是从主存区的命令参数开始执行。在高级语言中，命令参数对程序的“main”例程来说，常作为参数提供使用。若不可能，对存取该命令参数提供了专用的

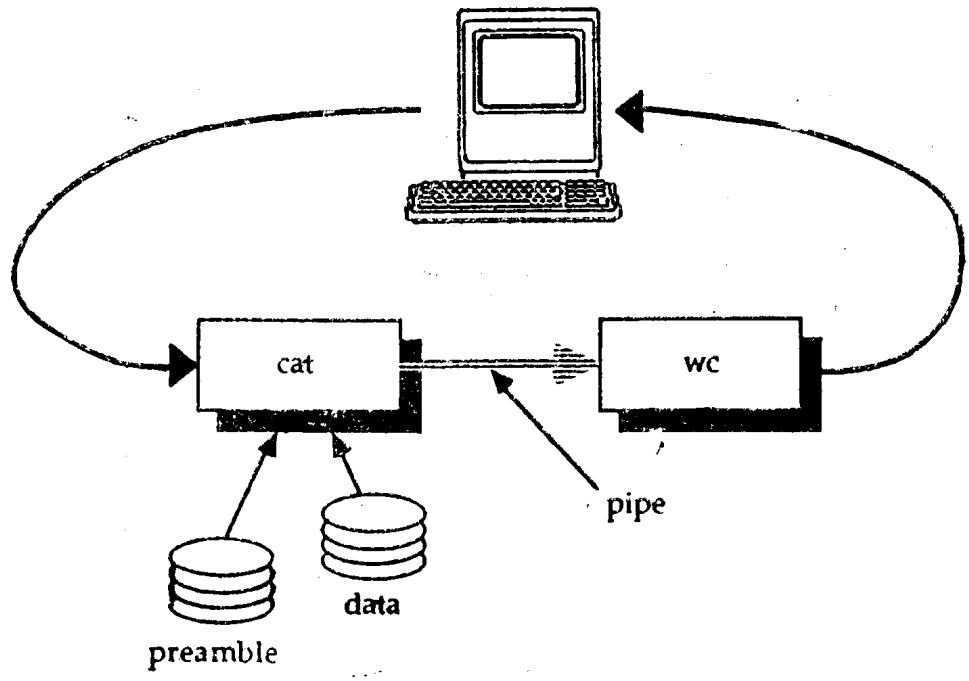


图1-3 管道线执行过程

库函数尽管命令行中的每个“字”是作为分立参数通过，各参数也仅仅是个正文串而已。在多数也仅仅是个正文串而已。在多数情况下，虽然shell需要把文件名的万用符（“wild cards”）扩充为多个参数，但shell不改变命令参数。shell还具有变量替换及引号字符的专用语法以防止参数随意变换。shell本身也处理I/O重定向，下面详述。因此，UNIX中的即使最简单的程序无需程序员费力就能“实行”这些特性。

UNIX执行各种各样的系统调用。每种UNIX系统提供的UNIX程序员手册的第2节把这些调用编写成文。系统调用提供的各类服务是：

·I/O操作。

这些服务提供了对共享I/O设备和描述其状态的全局数据结构的访问。它们打开和关闭文件和设备，读和写数据，置设备状态及读写系统数据结构。

·进程控制。

系统调用允许进程控制本身的执行。进程能在内存定位，设置它的调度优先权以及其它参数，在内存中锁住自己，装入和执行一个新的程序及等待事件发生。进程还能创建新进程。

·进程间通信。

系统调用允许一个进程向另一进程发送信息。进程间发送信息有许多方法：信号，管道，共享内存，报文队列以及信号灯。除了这些相当标准的进程间通信方法外，RTU执行异步系统自陷（trap）（AST）。这种自陷与信号很相似，但它能删除许多有毛病的限制信号。

·计时服务。

进程能使用系统计时和间隔时钟与系统操作同步。

·状态信息。

许多系统调用都返回关于进程及其子进程的状态信息，关于文件系统的信息，关于I/O设备的信息。

当使用高级语言编写程序时，系统调用执行起来就象专用库调用。因为许多系统调用都要返回某类型的值，所以系统调用通常都有一套用来说明出错的非法值，使用一个具体的变量存储错误代码，该代码就代表了已发生的某种错误。

·I/O系统调用

通常，对文件执行任何操作之前，该文件必须先打开。这允许操作系统去初始化各种内部数据结构以使后面更有效地执行I/O操作。字文件打开时。操作系统返回一个文件描述字（file descriptor），这是一个小整数，标识文件已打开。其它I/O操作把文件描述字当作一个参数而不是路径名。所以，文件名只被查一次。

这儿有两个打开文件的系统调用，它们是：

```
fd=Open (file, mode)
```

这儿，file是表明要打开文件的路径名的字符串，而mode是含有各种不同标志位，指出什么样的操作可以在打开的文件上执行。mode有许多值，有以只读方式打开已存在文件，有以只写打开已存在文件，或既读又写的方式打开已存在的文件。

另有一种打开文件的系统调用，是创建一个新文件或复盖一个已存在的文件（将原先文件中的全部内容替换掉）：

```
fd=create (file, protection)
```

同样，file是表明要建立的文件的文件路径名的字符串。Protection是数字。指明哪类用户可以打开文件进行各种访问。注意，最新UNIX版本允许用Open系统调用来创建文件，该系统调用指出了正确的方式位并把Protection作为附加的参数。无论怎样，create提供了与已存在程序的兼容性。

一旦文件打开，在文件上能执行的主要操作就是从文件里读出数据或往文件里写进数据。系统调用：

```
actual=read (fd, address, count)
```

其中，fd是原先打开的被读文件的文件描述字，从该文件中读出countte个字节数据到内存的address地址处。read返回值是实际传递的字节数。若返回值小于零，则读文件有错；若返回零值，指明读到文件尾而不是错。否则，读出的实际字节数通常是等于请求的字节数。如果遇到文件尾，actual可能小于count，对于fd指明引用的是特殊文件(设备或管道)时，actual小于count的情况也会发生。例如，关于终端设备驱动程序一般不读多行数据，它不管请求的字节数是多么多，一次只读一行。

系统调用：

```
actual=write (fd, address, count)
```

除了是往文件中写数据外，其它全和read动作一样。尽管设备驱动程序可选择往文件中写小于要求的数据，几乎所有的程序都将这种情况作为出错解释。

I/O操作通常是顺序进行的，所以下一个read或write调用将访问文件的下一个数据块。用下面的调用处理读、写要求，可以用任何顺序存取数据：

new_location=lseek (fd, offset, whence)

这儿，whence说明，offset字节数是从文件头开始还是从文件尾开始，或是上一次读写文件的最后字节处开始。lseek系统调用，返回从文件头计算字节数的那个文件的新的“查找指针”的新位置。

当不再需要打开的文件时，用下面的调用关闭文件：

result=close(fd)

若关闭成功，则result=0。一般来说，除非fd说明的文件描述字在前边未被打开，close都是成功的。

关闭文件描述字之后，企图还使用该文件描述字去读写必将失败。然而，也能重新使用那个文件描述字，必须在后来又有Open和Create调用时，就可能返回和前一次read和write调用的相同的文件描述字值。这一点儿，对I/O重定向十分有用。每个程序都是以几个已打开的文件描述字开始执行。由命令解释程序执行的程序，这些文件描述字是：

·0. standard input文件。通常是终端，但可以通过I/O重定向，如用<file来改变。

·1. standard output文件。同样，通常是终端，但也可以通过I/O重定向如>file来改变。

·2. standard error文件。同样是终端但可通过I/O重定向改变输出方向。语法要求I/O重定向随不同的shell而变，但2>file重指标准UNIX shell中的标准出错文件。

一般来说，命令解释程序通过首先关闭这些标准文件描述字之一而后打开为读写所指定的文件完成I/O重定向。实际情况稍复杂些，一旦关闭了自己的标准输入文件，命令解释程序将不能读多个命令。为避免这个问题，命令解释程序首先用fork系统调用建立一个新进程：

pid=fork ( )

建立新的进程之后并在新旧进程中返回不同的值，最新建立的进程叫子进程（child），原进程是父进程（parent）见图1-4。父进程能执行超越子进程的特权控制。实际上，父进程可用系统调用：

pid=wait (status)

等待子进程结束执行。

这儿，pid标识几个子进程中哪一个结束了执行。status参数值是子进程结束执行，返回时的理由。进程可能自动地和在status中说明一完整代码而退出。进程也可能作为出错结果或从终端或从一进程接收到信号而结束执行。

对fork系统调用的返回来说，子进程几乎和父进程一样。但当pid是父进程中最新建立的子进程的进程ID时，子进程中的pid将为零。父进程和子进程都能从相同的文件描述字中进行读和写而且具有相同的最初内存内容。然而由于子进程是父进程的一个复本，所以子进程能够在不影响父进程的情况下，关闭文件描述字并打开新的文件。父进程和子进程各自也可以不影响其它进程的情况下修改其自己的内存。

打开文件描述字不是彼此的完全复制，也不是连续地共享某些信息。实际上，复制在父进程和子进的文件描述字共享相同的搜索指针。父进程和子进程往输出文件上写数，不用复盖由其它进程写过的数据，这种方法的写，变得很容易。当保证数据被读一次的情况下，还

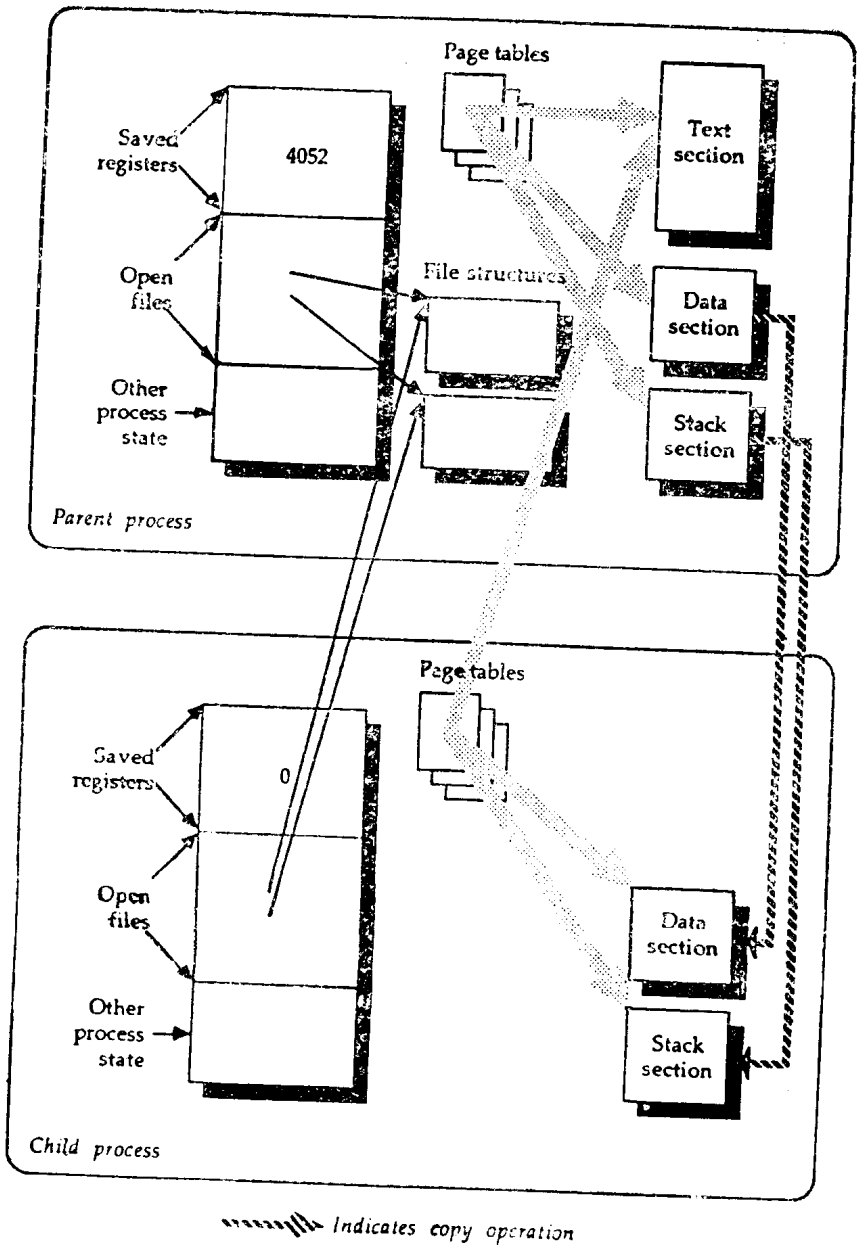


图1-4 fork操作

允许父进程和子进程去读相同文件的数据。

一个程序使用：

```
new_fd=dup (old_fd)
```

也可以生成任意打开文件描述字的复本，这个关系式说明，将把由Old_fd指明的文件描述字考贝到不是当前使用的最低编号的文件描述字。

任何进程可能执行由如下的系统调用的新程序：

```
result=exec(program, argument0, argument1, argument2, …)
```

这儿，program是要执行的命令或程序的路径名：argument0，argument1，…都是程序的参数。根据约定，argument0是程序本身的名字。参数列表由0结束，则零不可能是有效的

字符串。

新程序替换了当前进程正执行的程序。若exec调用成功，exec将在读出（sense）中没有返回值，即对那个后跟的第一个程序的exec调用语句将不能由该进程执行。然而，可能在由建立的新进程的子程序调用中获得执行程序的结果，以便执行程序并等待进程退出。

命令解释程序执行I/O重定向使用的实际顺序是：

1. 打开I/O重定向中指定的文件。如果打开不成功，即出一出错报文和返回并等待另一个命令行。

2. 为建立新的进程，调用fork。

3. 父进程应先关闭为I/O重定向打开的文件描述字，然后调用wait。

4. 子进程将根据I/O重定向是标准输入，标准输出，标准出错的实际作用，调用带有0，1或2的参数的close。

5. 子进程为引用为重定向打开了的文件描述字，使用dup建立标准输入，标准输出或标准出错，调用dup之后，该文件描述字可能被关闭。

6. 最后，子进程执行命令行指定的程序。

与系统调用相关的其它文件系统，多数与我们无关。比如，link和unlink调用是建文件的新名字或删除文件名，这些无需调用设备驱动程序。同样，stat和fstat调用返回由路径名或打开的文件描述字指明的有关文件信息，也是无需调用设备驱动程序（该信息非常象用ls列长格式信息）。

但是，设备驱动程序的一个非常重要的系统调用是ioctl（I/O控制）。这是除了读写数据之外的综合（catchall）进行I/O控制的功能。

调用格式简单但易使误解：

```
result=ioctl(fd, command, arg)
```

这儿，fd是常用的标识已打开的文件描述字；command说明要执行的操作；而arg是与命令有关的信息。arg常是内存控制块的地址，作为ioctl的操作参数的还可能是一个随机的数据量。

据约定，脱离命令解释直达设备驱动程序，不同的设备驱动程序除非是驱动同类设备不同场合的设备，是不能执行相同命令，例如，终端设备驱动程序执行公共的命令集，该命令允许程序改变通信线的数据速率或者改变返屏什么样的字符。磁带设备的设备驱动程序执行不同的命令集，一是反绕磁带的命令，二是跳到磁带的下一个文件的命令。但当程序询问终端设备反绕之事时，终端设备驱动程序就将产生一个错误信息；同样，当程序对磁带设备请求停止回声（返屏）时，磁带驱动程序也产生出错信息。

1.4 进程控制和调度

本节讨论用户进程是如何形成，如何表示其状态，如何调度其执行。

进程建立和删除

在最初的系统引导过程中，建立了两个基本进程：

- 进程0　交换程序进程（swaper）
- 进程1　初始引导进程（init）

只要UNIX系统一激活或调用，这些进程就存在交换程序支持存储管理操作。初始引导

进程负责（直接或间接）启动全部其它用户进程。其它一些由系统建立的进程是用来为进程的活动服务的，如页面调度。

这些进程建立之后，由fork系统调用建立用户进程。当用户进程调用fork时，核心建立一个新进程，该新进程与调用进程几乎是一样的，所不同的是新进程有一个新的唯一的进程标识号（PID）和一个新的父进程标识符（PPID）。原先进程叫作父进程，新建的进程叫子进程。

用户进程通过exec调用，用新的代码映象替换正在执行的代码映象。为建立新进程运行新程序，现存的进程必须先调用fork然后再调用exec.在系统启动时,init进程用fork建立了几个附加进程，这些附加进程是以/etc/inittab文件文本为依据、详见UNIX程序员使用手册中init（8）中的完整描述。一般情况下，init建立几个后台进程（deamons）和其它的进程gettys，gettys是监控用户注册每一个交互对话的终端。

当用户注册到系统时，监控终端的进程通常要多次调用exec，以便运行部分注册序列的不同程序。最后，为从终端上读取用户命令，必须装入shell或命令解释程序的程序。shell能在它自己环境上输出某些命令，但shell必须频繁地运行另一个程序（即执行已知映象）。为此，shell用fork建立一个新进程，然后使用exec装入和执行所需的进程。父shell进程不是在新进程正运行时等待就是继续读和执行其它的命令。当由shell启动命令进程完成其工作时，就调用exit，然后继续存在下去。当shell接收注销（logout）命令时，它也调用exit。这时，一直等待shll退出的init进程用fprk建立新进程gettys监控终端接收下一个注册请求。

进一步讨论表明，任何时刻存在的用户进程数目是不定的。对每个交互终端至少有一个进程，终端命令解释进程频繁地建立附加的进程响应各种命令。那些进程可以顺序地用fork建立附加进程。

进程定义结构—user和proc

核心为每个进程维持两个可描述的数据结构：

- user，含有关于进程激活时所需进程的全部信息。u是当前活动进程的User数据结构.驱动程序频繁使用User结构中的许多字段。
- proc，含有那些不活动进程或是已被交换出内存后所需的进程信息。设备驱动程序不能显式地访问proc结构中的字段，但它们经常在调用核心子程序时，作为参数，将指针传递给该结构。

这些数据结构分别在/usr/include/sys/user.h和/usr/include/sys/proc.h中定义。

当一个进程被交换到盘上时，其user结构也随之写出去。全部进程的proc结构保存在数组中通称进程表，该进程表常驻内存。用户进程的状态全部由这两个结构的数据定义。

进程调度，页面调度和进程交换调度。

一般来说，多用户进程存在着竞争CPU，内存和其他硬件的使用。RTU核心通过执行进程调度，内存页面调度及映象交换来安排这些竞争的代码。在任何给定的时间中，有许多用户进程就绪等待运行。在核心中的调度例程搜索进程表并选中就绪的又具有最高优先权的进程，该进程便获将了CPU的控制权。调度程序随时执行这种选择。每一个用户进程，最终都能分享执行时间。

因为RTU是要求具有调页的虚拟系统，由核心和活动的用户进程占领的全部虚地址空间