目　录

UNIX 操作系统分析报告

中国科学院计算所　刘日昇　　孙玉方

序：美国 Bell 实验室 D. M. Ritchie 和 K·Tompson 在 PDP11 系列机上研制的UNIX 系统是当前国际上公认的相当成功的一个分时系统。

整个系统包括C 编译程序、命令解释语言 Shell 、用于软件开发、资料准备等各种各样的实用程序，而其核心部分是 UNIX 操作系统。该系统目前已成为 PDP11 、VAX11、Interdata8/32 等小型机系列，及 M68000、Z8000、Intel8086 等高档微型机的主要操作系统。

由于 UNIX 操作系统采用了一系列成熟技术和精巧算法，所以短小精悍(整个操作系统的代码在 10000 行左右）而功能很强（在许多方面可以和“大型”系统媲美），成为人们学习和研究操作系统的一个理想对象。在美国和澳大利亚等国家，它是大学计算机科学系操作系统课程的主要教材和参考书。有人评论它是“操作系统中的 Fortran”。“就如 CP/M 是 8 位微型机的标准操作系统一样，UNIX 将是 16 位微型机的标准操作系统”。有的甚至认为它已远远超出操作系统范畴，而是继高级语言兴起阶段，结构程序设计阶段后出现的自动程序阶段的成功代表作。这种观点很值得我们重视。在我国有不少科研部门、学校和生产单位也在学习，使用和移值UNIX 系统。

1980 年 4 月至 6 月，在仲萃豪导师指导下，我们学习了 UNIX 操作系统源代码及《注释》等有关资料。以后我们又按模块化层次结构方法重新构造了这个操作系统，并在此基础上完成了硕士论文及其它有关论文和讲义。同时，我们在若干研究所、高等院校介绍了 UNIX 操作系统。为了满足广大读者的需要，根据教学实践，我们重新编写了这份《分析报告》。

在本文形成过程中得到了仲萃豪导师的指导，蒋士骕和高庆狮教授给予了热情鼓励，范植华和陈修同两同志参加了 UNIX 操作系统分析的早期工作。对于他们的支持谨致谢意。

前　言

UNIX 操作系统是一个通用的，交互式的分时系统。它是美国 Bell 实验室的 D. M. Ritchie 和 K. Thompson 于 1969～1970 年首先在PDP-7 上实现的，1971 年在 PDP-11 机上运行。1974 年 7 月公布于 C. ACM 杂志上。自 UNIX 系统运行以来，由于它的简单，通用，有效和使用方便，得到了越来越广泛的应用。到 1979 年底在世界各地已有 2000 多个装置运行了这个系统。UNIX 取得了如此大的成功，以至于 Bell 实验室把“UNIX”这个名字作为它的商标。

UNIX 系统自产生以来，经过了多次修改，产生了很多版本，但它的根本思想没有发生变化。目前已产生了第七版。现今最流行的是第六版。此外，在 UNIX 操作系统的基础上，人们又发展了许多其它种类和用途的软件系统，如 MERT 实时操作系统，网络 UNIX，微处理机 UNIX，小型卫星机系统，以 UNIX 为核心的关系数据库系统 INGRES 以及程序员工作台 PWB/UNIX 等。

UNIX 系统吸取了当时许多操作系统的成功经验，例如美国 MIT 等单位研制的 CTSS 和 MULTICS 系统等。它删除或改造了这些系统中与基本功能关系不大的部分，吸收了它们的优秀成果，大大压缩了系统的规模，从而使得它能以少量的代码，在一台小型计算机上完成许多大型操作系统也不多见的功能。

目前，UNIX 系统的影响已经不再局限于操作系统这一范围。在国外、国内的计算机科学与工业界正在出现一种新的观点，认为目前软件发展已经达到了第三阶段。第一阶段是高级语言兴起阶段，这是从 60 年代初期开始的，从 FORTRAN 和 ALGOL 60 开始到 ALGOL 68 为止，这一阶段主要是通过高级语言和编译系统提高程序自动化。第二阶段是结构程序设计阶段，从 70 年代初开始，从PASCAL 到 ADA 为止。在这十年中人们主要关心的是软件可靠性，由此引起对程序设计方法和程序语言结构的研究。ADA 语言的出现是这一时期的总结。第三阶段是自动程序阶段，这一时期的特点是建立由若干层程序语言，程序工具和程序设计方法构成的大型软件系统，使得人们可以利用各种手段方便地编制出可靠的程序。这种系统叫做程序工具环境。它是 20 年来有关程序自动化理论与实践的总结，是在目前条件下所能达到的，关于程序自动化的最先进阶段，而 UNIX 系统则是已经研制成功的这样的系统的典型代表。

UNIX 系统的主要特点有如下几方面：

1．具有分级结构的，可装卸文件卷的文件系统。整个文件系统形成从根目录表开始的树形分级结构。用户可以把自己文件卷上的文件系统连入原有的系统上。以后也可以把它拆卸下来。

2．文件、目录表和外部设备作为文件统一处理。文件无结构（无记录），无类型概念。所有文件均作为无格式的字符流序列。这给用户提供了一个简单，划一的接口。

3　按照用户的要求，系统可动态地产生和消灭子进程。用以执行用户的命令。系统也提供用户的进程之间简单的通讯功能。

4．系统提供功能完备而使用灵活的命令程序语言，即 Shell 语言。这是 UNIX 系统和用户的接口。它既是与终端用户交互作用的命令语言，又是在命令文件中执行的程序设计语言。用户通过这个 Shell 语言使用 UNIX 系统的各种程序设计工具。

5．UNIX 提供十几种程序设计语言和其它大量子系统。如 C, FORTRAN 77, BASIC, SNOBOL, APL, ALGOL 68, PASCAL, 汇编程序，正文编辑程序，连接装配程序，公式排版程序等。

6．UNIX 的操作系统核心以及外层所有程序中，绝大部分是用C语言书写的。这是一种不太高级的程序设计语言，但使用方便，程序紧凑，效率高。因而大大方便了 UNIX 系统的阅读，修改和移植。

这份源程序假定的模型系统的硬件配置如下：

PDP 11/40 中央处理机
RK05 磁盘
LP11 打印机
PC11 纸带输入/穿孔机
KL11 终端接口

一、系 统 概 述

1.1 程序结构

UNIX 核心源程序（指我们分析的这份源程序，下同）划分为 44 个源代码文件。这些文件按照编译方式的不同分为三类，它们分别用文件名字后面的一个字母表示。其中 s 表示汇编代码文件，c 表示C语言程序文件，h 表示C语言的全局变量文件。汇编代码文件有 2 个。C语言程序文件有 28 个。它们可以独立编译，然后通过连接装配程序组成一个目标程序。C语言的全局变量文件有 14 个。它们不是独立编译的，而是附属于某个C文件一起编译的。

这 2 个汇编语言文件分为 33 个汇编子程序，共占约 1000 行代码，它们主要用于系统初启，中断处理，用户虚拟空间与核心虚拟空间之间数据的传送以及提高运行效率等等。C语言程序的 28 个文件分为 190 个子程序。所以 UNIX 核心源程序共分为 223 个程序模块。

核心程序的结构基本上是以全局变量为中心的无序模块结构。文件作为独立编写，编译的程序单位，但它们之间的调用关系是非常复杂的。因此，在本文下面各部分介绍源程序的各个部分时，一方面要遵照源程序的次序，另一方面尽量考虑到层次依赖关系和概念的完整性。按照系统功能的划分，各部分介绍的次序如下：

地址映射与存储分配
进程调度与交通控制
进程映象管理
中断，捕俘与系统调用
软中断与跟踪管理
块设备管理
文件系统
字符设备管理

在UNIX 核心程序之外运行用户态的程序，这除了用户程序外，主要包括了大量的应用程序和各种子系统。它们全都在 Shell 命令解释程序的管理之下，作为进程在处理机的用户态下运行。它们只能通过由核心提供的 41 个系统调用程序请求核心服务。而用户则只能通过 shell 语言请求 UNIX 系统在终端上或后台得到服务。本文只介绍核心源程序。

下面再介绍C语言子程序（简称C程序）之间的调用，返回过程。

C程序之间连接接口由编译程序完成，它在每个C程序入口插入入口子程序（csv）调用指令，而在出口处插入出口子程序（cret）调用指令。C程序的连接过程和运行时栈的变化如图 1.1 所示。

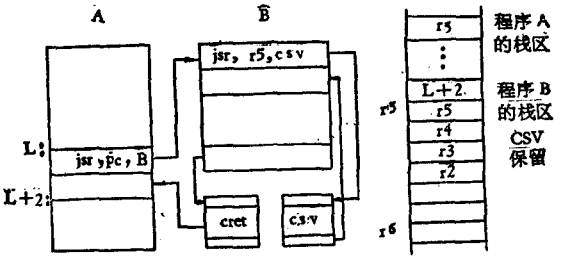


图 1.1 C程序的连接

考虑程序A调用B的连接过程。程序A调用程序B时把返址 L + 2 填入B的栈区栈顶。程序B的入口指令调用 csv 时把 r_5 进栈。csv 顺序地将 r_4、r_3、r_2 进栈。r_5 指向栈内刚刚保留 r_5 的单元。而被保留的 r_5 又指向上一栈区（程序A的栈区）内保留原 r_5 的单元。以此类推，程序栈中保留 r_5 的单元组成一个链，寄存器 r_5 指向这个链的链头。每当要退出一层程序时，r_5 就可以用来找到被保留的 r_4、r_3、r_2 和返回地址。因此 r_5 又叫做环境指针。而 r_6 总是指向栈顶，又叫做栈指针。r_4、r_3、r_2 用来作为被调用程序的局部变量，例如指针。r_1 和 r_0 不保留，用于调用程序和被调用程序之间传送参数。

汇编语言子程序不遵守上述的连接约定。

1.2 系统功能

1.2.1 存储管理

UNIX 系统使用 PDP 11/40,45,70 机器上的地址映射部件。对于 11/40 来说，它具有两套地址映射寄存器，分别用于核心态程序和用户态程序。每套寄存器分别为用户程序和系统程序建立各自的虚拟地址空间。每套寄存器含有 8 对映射寄存器，每对寄存器表示内存一块连续区域的起始地址和长度等特征。这样一块存储区叫做段，按照 PDP 的术语也叫做"活动页"或简称为"页"。每个虚拟空间最多可达到8 页，每页最大为 8192 字节或 4096 字。可见，PDP 11 的页面管理，既不是通常的页面管理方案，也不实现虚拟存储器。

存储器的分配分为内存，磁盘两种。内存分配指的是内存中除 UNIX 核心以外的内存部分。磁盘分为两部分：一部分用于程序对换，另一部分用于文件存储。内存和盘对换区的申请和释放采用同样的算法，即"最先满足"算法。两类资源各有一张可用表，每个表目指向一块连续的可用存储区。内存申请单位是 64 字节(32 字)，磁盘对换区申请单位是 512 字节(256 字)。用于文件存储的磁盘部分由文件系统管理。

1.2.2 进程管理

在 UNIX 系统中"进程"这一概念可以粗略地定义为"程序的执行"。

系统初启之后，只有进程 0 和进程 1。进程 0 负责程序对换和分配处理机，在核心态下运行。进程 1 为每个终端用户建立一进程，而每个用户进程又可任意生成子进程。因此，进程 1 成为系统中除进程 0 外所有进程的祖先。

系统核心的所有程序都是作为进程的一部分而执行。当一进程在用户态运行时，执行的是用户程序。此时若发生中断或捕俘，则转入核心态，执行核心程序。进程在核心态和用户态执行的程序分别使用各自的存储映射机构，各自的栈和栈指针。在 UNIX 系统中，有时把进程 i 的用户程序部分叫做用户进程i，而把其中的核心程序部分叫做系统进程 i。这里唯一的例外是，进程 0 只有核心态部分。其主要部分是程序 sched，它没有用户态程序。

进程存在的唯一标志是在 proc 表中占有一个表目，每个表目也叫做 proc 结构。proc 结构中含有该进程的随时可用的状态信息。proc 表有 50 个表目，常驻内存空间。关于进程的更多的状态信息是仅当该进程的映象送入内存时才使用的。这部分信息就叫做 user结构，它与该进程的系统栈一起构成"进程数据区(ppda)"，在物理上它与该进程的其它数据连在一起，同时换入、换出内存。然而，ppda 处于核心地址空间，用户程序是无法存取的。

一个进程可借助系统调用 fork 建立子进程，子进程自动继承父进程的正文段，数据段和打开文件。但是，为了建立多样化的子进程，这个子进程可用系统调用 exec 执行一个文件，从而更换正文段和数据段。父进程可用系统调用 wait 等待子进程结束，子进程可用系统调用 rexit 请求父进程结束自己。

1.2.3 进程通讯

系统进程借助 sleep 和 wakeup(或setrun)实行同步。它们是不带信息的，比 P、V 操作更低级的同步工具。sleep 使当前进程以指定原因和优先数睡眠，而 wakeup 则唤醒在指定原因上睡眠的所有进程，用户进程不能使用这些工具。

UNIX 的调度策略规定，在核心程序执行期间发生中断或捕俘时，不发生进程的转换。仅当用户程序执行过程中被中断(或发生捕俘)并且在处理中断之后返回时，才可能调用低级调度程序转让处理机。因此，在系统程序执行期间，只要它自己不调用 sleep(或 swtch)主动要求放弃处理机，就不会在它的代码中任意插入其它进程的代码。这就大大方便了核心程序内并发进程的管理。

但是，在核心程序内会插入中断或捕俘处理程序。它可能改变进程的状态或修改公共数据。因此，核心程序还是要提防这种意外的干扰。这只须提高某段代码的处理机优先级以屏蔽某些类别的中断即可。

在用户程序一级，一个用户的诸进程之间可利用核心提供的软中断(信号)实现某种通讯。所谓软中断，即一个进程在自己或同一终端上的其它进程的 proc 结构中的一项 (p-sig) 设置一个从 0 到 19 的整数。在被设置信号的进程某个时候以后，如退出捕俘处理程序，退出时钟中断程序或执行 sleep 进程时，便检查此标志设置与否。如设置，则按指定号码执行自己预先规定好的程序。

为了使父进程对子进程实行跟踪，UNIX 系统把上述软中断功能加以扩大，使得每当被跟踪进程检查软中断时插入父进程采取的一系列跟踪行动，这包括两进程之间少量信息传输。

为了在同一用户的两个进程之间传送大量信息，可利用文件系统的 pipe 功能，建立一个无名的，先进先出的临时文件，分别对它们进行有控制的读写，从而传送大量信息。

UNIX 系统的进程通讯功能的能力是比较弱的，这为核心程序内并发活动的管理和控制带来复杂性。这是 UNIX 系统的一个主要弱点。

1.2.4 中断、捕俘与系统调用

在 PDP 11 系统中，处理机的优先级为 0 到 7 级，这由处理机状态寄存器 PS 的第 5—7 位表示。一般情况下程序在 0 级下运行。当中断发生时，CPU 把当前处理机状态字 PS 和程序计数器 PC 存入当前栈，并以固定单元的值装配 PS 和 PC，从而改变了优先级。中断是外部设备请求 CPU 服务的一种方式，其中断优先级可以为 4、5、6 级，对应的中断处理程序优先级开始与之相同，以后可用程序修改。仅当中断优先级高于当前处理机优先级时，才会发生中断。

在 UNIX 系统中，没有设立外设管理进程。外设的管理(如缓冲区、队列和启动)，是由使用它的进程调用有关子程序完成的。因此，外设的中断处理程序往往要进行出错处理，队列操作以及与调用进程进行通讯。

捕俘来自 CPU 内部原因分为 11 种。其进入与退出方式与中断处理相同，因而公用一段汇编代码。但它的处理方式，除了少数当场处理外，大部分将它们转化为对应的软中断信号，向进程自己发送。然后，在处理这个软中断时便转到用户提供的处理程序处理。在 11 种捕俘中有一种是由 trap 指令引起的，这是系统调用的引导指令。该指令的低 6 位用于指出系统调用程序的入口。

在本系统中提供了 41 种系统调用，这是用户程序与核心程序交往的唯一界面。它为用户程序读或写后者不能存取的核心数据，以及提供有关进程，文件管理方面的功能。

1.2.5 输入输出管理

输入输出(I/O)系统主要分为两类：面向块的 I/O 系统和面向字符的 I/O 系统。前者如磁盘、磁带等，后者如纸带输入/穿孔输出机、打印机和终端设备。

块 I/O 系统通过一层缓冲软件管理。为此，系统设有一缓冲池，内含 15 个缓冲，每个缓冲 512 个字节。当某台块设备要读取数据时，便先查该数据是否在该设备的缓冲队列中。如在，则直接读出。否则申请一个缓冲，读入所需数据，链入缓冲队列中。写的过程也类似。这样，减少了 I/O 传输次数。实际上，这些缓冲池和块设备一起起着虚拟存储器的作用，当然这完全是用软件实现的。

字符 I/O 系统是通过字符缓冲实现的。由于字符设备速度较慢，在输入输出期间还要进行字符加工，因此利用了较小的缓冲区。每个缓冲 4 个字(8 个字节)，其中 6 个字节用于存放字符，一个字用作队列指针。每类设备都有一个至三个缓冲队列。系统通过管理这些缓冲队列以实现有效的传输。

在块设备的使用中，还有两种是不通过缓冲区的，一种是对换系统使用的。这时块设备中的对换空间与用户内存区之间以 32 字为单位传送。另一种是原始磁盘管理，是系统内部检查使用的，它以字节为单位访问磁盘，相当于访问字符设备。

终端输出也有一种无缓冲的，直接的使用方式，用来向控制台上的操作员报告初置或诊

断消息。

1.2.6 文件系统

UNIX 系统有三类文件：普通文件、目录文件和特别文件。

普通文件是无结构的字符串文件，没有记录概念。文件长度允许动态增长。对文件的存取无顺序和随机之分。每次读、写文件的一个字符之后，文件指针便指向下一个字符，但用户可用系统调用 seek 重置指针位置，从而实现随机存取。

目录文件与普通文件一样管理，系统中除一个根目录项外，任一文件都是某一目录文件的下级，由此形成文件系统树形分级结构。树的端点是普通文件，其余为目录文件。然而违背树形结构的一个例外是，一个普通文件可由两个以上目录登记项所指向，称之为联结。

UNIX 文件系统的另一特征是把 I/O 设备也作为文件处理，称为特别文件。对此文件的读、写只要求启动相应的设备驱动程序。

文件系统为用户提供可装卸存储卷的功能。文件系统除了有一个根设备（整个系统的根目录在这里）以外，每个用户还可在自己的存储卷上建立一独立的文件分级结构。利用系统调用 smount，用户就可以把他自己的文件树结构的根目录连到一个文件树的某一末端（空）文件上。以后对此末端文件，即普通文件的存取，便代之以对存储卷根目录的存取。于是存储卷上的文件就如同根设备上的文件一样存取了。

在用于文件存储的磁盘存储器的第 1 块专用块（superblock）里含有一个由 100 项组成的索引表，用来管理文件存储空间里所有的空闲块。这些块按 100 块一组划分，每一组的最后一块用来保持上一组 100 块的地址。最后一组不多于 100 块，它们的地址存在专用块的索引表中。系统通过这种成组的拉链表，可以对任意多的空闲块实行后进先出管理。

分级目录表的实现采用了 i 节点表的方法。文件系统中的每个文件的特征信息都分别保存在 i 节点表的相应表目中，而在目录登记项中只需存放文件名字和对应的 i 节点（在对应盘上 i 节点表中）的编号。与文件有关的操作均通过对应的 i 节点进行（详见第五部分）。

二、存储管理与进程管理

2.1 硬件寻址机构

这里介绍的模型系统采用 PDP11/40 中央处理机，但也适用于 PDP 11/45,70。

PDP 11/40 处理机有 9 个通用寄存器，字长 16 位。在 UNIX 系统中它们的作用如下：

r_0、r_1：传递过程调用的输入，输出参数。

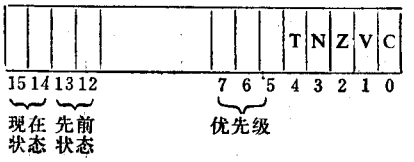
r_2、r_3、r_4：用作过程的局部变量。

r_5：环境指针。

r_6(sp)：栈指针。用户态，核心态各有一个。

r_7(PC)：程序计数器。

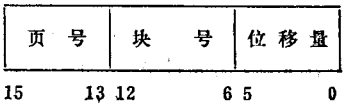
处理机状态寄存器(PS)结构如下：



状态位的解释是，00：核心态；11：用户态。第 5、6、7 位表示处理机优先级。第 0 位(C 位)为进位位，第 1 位(V 位)为溢出位，第 2 位(Z 位)为零位，第 3 位(N 位)为负位，第 4 位(T 位)为捕俘位。系统调用执行有错，则 C 位置 1 (见 3.4.1 节)，软中断处理结束，则 T 位置 0 (见 3.6.4 节)。

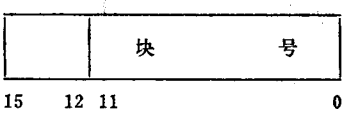
处理机寻址采用页式管理方案。然而这仅仅是为了使用户的地址空间(虚空间)和内存的存储空间(实空间)相分离，使得用户的程序和数据在内存空间中按需要上下浮动。这里的虚空间不是指内、外存一致看待的一级存储器。而且这里所说的“页”也不是指通常虚拟存储系统中作为物理传输单位、大小固定的“页”。在 UNIX 系统中页的大小是可变的，以块为单位(每块 64 字节)，最大可达 128 块。处理机在用户态和核心态各使用一组存储映射寄存器，把程序中虚地址映射为内存空间实地址。PDP 11/40 每组映射寄存器包括 8 对寄存器，每对寄存器对应用户空间的一页。每对映射寄存器由一个地址寄存器和一个对应的说明寄存器组成。地址寄存器指出该页在内存中的起始块号，说明寄存器指出该页的一些特征，如该页大小，存取控制特征等。

用户的虚拟地址组成如下：



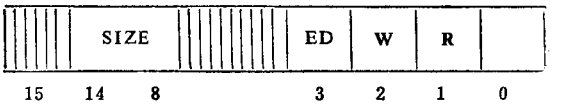
地址的高三位指出页号，用于找出 8 对映射寄存器中的一对。而其中的地址寄存器中指出的块号与虚拟地址中第 6 至 12 位指出的块号之和即内存块块号，第 0 至 5 位的位移量指出块内字节号。这个块号与块内字节号并在一起形成对应的内存地址。

地址寄存器的结构如下：



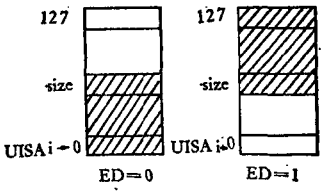
地址寄存器在 PDP 11/40中只用低 12 位，其寻址范围可达 2^{18}(即 256 K)字节。

说明寄存器的结构如下：



第 8 至 14 位(SIZE)共 7 位，表示该页边界块号，即最大块数为 $2^7=128$。第 3 位 (ED 位)为扩展方向指示位。其意义如下图所示。假定地址寄存器 UISAi 指向一最大页(128 块)的第 0 块，如 ED 位为 0，则表示此页从 0 块到 SIZE 块有效。如 ED 位为 1，则表示此页向下增长，就是说此页从 127 块向下扩展到 SIZE 块有效。在通常情况下 ED=0，当程序虚地址的块号大于 SIZE 时，发生越界错误，如

ED=1，则当程序虚地址的块号小于SIZE时才表示越界错误。在UNIX系统中这一特征位用来表示向下扩展的用户栈(见2.3节。)



在PDP 11/40中8个核心空间地址寄存器地址为KISA 0到KISA 7,说明寄存器为KISD 0到KISD 7。用户空间地址寄存器为UISA 0到UISA 7,说明寄存器为UISD0到UISD7。图2.1举例说明了用户空间虚、实地址的转换。

在PDP 11/45和11/70中每个空间有16对映射寄存器，其中前8对用于映射指令(I)空间，后8对用于映射数据(D)空间。

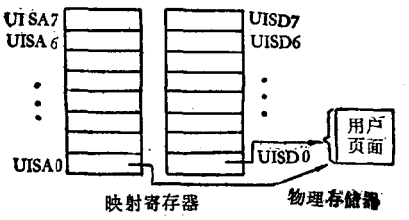


图 2.1 用户地址空间映射机构

2.2 进程与进程映象

在UNIX系统中，进程可粗略地定义为“执行中的程序”。这个定义未能很好地适合进程0和进程1(初始阶段)的情况。但它反映了进程要做的事情。在操作系统中，对进程这一概念可以从两个方面加以研究。

在较高级的方面，进程是一重要的组织概念。可以把计算机系统看作是若干进程组合的活动。进程是系统中活动的实体，它可以生成和消灭，申请和释放资源，可以互相合作和竞争。而真正活动的部件如处理机和外部设备则看不见了。

在较低级的方面，进程是不活动的实体，而处理机则是活动的。其任务是对进程进行操作。处理机在各个进程映象之间转换。

我们现在关心的是低级方面的解释，即进程的映象结构，执行细节以及进程之间的转换手段。

在UNIX环境下，进程有如下特征：

1．在proc数组中占有一非空表目，在其中保留相应进程的状态信息。

2．每一进程有一“进程数据区”(ppda)，保留相应进程更多的信息和系统栈。

3．处理机的全部工作就是执行某个进程。

4．一进程可生成或消灭另一进程。

5．一进程可申请并占有资源。

进程映象是一抽象数据结构，是伪处理机的当前状态。进程映象在物理上可分为三个不同的部分。

① proc结构，是数组proc中的一个元素。因proc数组常驻内存，所以它总是可以存取的。

② 数据段，其中分三部分：底部是“进程数据区”(ppda)，1024字节。其中的底部289字节为user结构，含有对应进程更多信息。其余367个字用作该进程的系统栈，从顶向下扩展。ppda在物理上是进程数据段的一部分，以便有效地与其它数据一起换入、换出内存，但它们却属于不同的地址空间。ppda是核心空间的第6页，即核心空间第6页地址寄存器(KISA 6)总是含有当前正在运行进程的ppda起始块号，也是该进程user结构的起始块号。而数据段的其他部分属于该进程的用户空间。所以进程的用户程序部分不能存取它自己的ppda。数据段的顶部是用户栈区，从用户地址空间第7页(最高页)向下分配。在物理上也是从顶向下扩展。数据段的中间部分是用户程序数据区，可读可写。其中也可以有非再入的程序正文。

整个数据段由进程0的sched程序换入或换出内存。

③ 正文段，包括可再入的代码或常数。

正文段可由其它进程共享。它也可能不存在。它在物理上不一定与数据段连在一起，不与数据段一起对换。

除了进程 0 之外的每个进程都在两种状态之一运行，即核心态和用户态。当一进程在用户状态下运行而发生一中断或捕俘时便进入核心状态。一个进程在两种状态下拥有不同的特权，有不同的地址空间，有不同的栈，有不同的栈指针(SP)。它们唯一的共同点是有共同的 proc 结构和共同的 user 结构。

由于一个进程内的这两部分差别这样大以致于在 UNIX 中把进程 i 的核心态部分叫做系统进程 i，而把它的用户态部分叫做用户进程 i。

各个系统进程的地址空间除了第 6 页映射到自己的 ppda 而不同外，其它 7 页的映射都是一样的，即如系统初启时装配的那样，前 6 页的虚拟地址等于实际地址，第 7 页映射到最高物理地址 4 K 字的专用寄存器上。这里说的虚拟地址指的不是内外存作为一级存储器的虚拟空间地址，而是指页式存储管理方案中的逻辑地址。

为了建立用户的地址空间，首先在每个进程的 user 结构内的两个数组 u_uisa 和 u_uisd 中分别保存用户空间地址和说明寄存器的样本。这是 estabur 程序根据给定正文段，数据区和用户栈的块数及指令，数据空间是否分离而装配的。因为进程的正文段、数据段的物理地址是随对换而变化的。所以样本地址寄存器内的地址假定正文段以第 0 块为基址，数据段以第 16 块为基址。因而在该进程换入内存后便可根据样本寄存器内地址加上正文段和数据段物理起址装配硬件地址寄存器，而说明寄存器不变。暂且把样本寄存器表示的空间称为样本空间。

进程的转换是在 swtch 程序中进行的。当一进程调用 swtch 时，它的 r_5、r_6 便保留在其 user 结构的 u_rsav 中。以后当 swtch 选中此进程重新运行时，便使核心空间第 6 页寄存器(KISA6)指向被选进程的 ppda 起址(后者保存在其 proc 结构的 p_addr 中，而且这时 ppda 与其数据段已换入内存了)。由于 ppda 起址也是 user 起址，故由此可找到 user 中的u_rsav 或 u_ssav，从中恢复先前保留的 r_5 和 r_6。同时根据 proc 结构中正文段和数据段地址信息装配该进程的用户空间映射寄存器，该进程就可以运行了。下面介绍三个重要的数据结构。

1. proc 结构。

char p_stat；定义进程的七种状态之一：

NULL：未分配；

SSLEEP：高优先权睡眠(优先数<0)；

SWAIT：低优先权睡眠(优先数≥0)；

SRUN：准备好运行(就绪)；

SIDL：进程生成；

SZOMB：进程结束；

SSTOP：因被跟踪而暂停。

char p_flag；6 个标志位组合指出进程特征：

SLOAD：第 1 位，在内存与否；

SSYS：第 2 位，调度进程(不能换出)；

SLOCK：第 3 位，不换出；

SSWAP：第 4 位，对换标志；

STRC：第 5 位，被跟踪标志；

SWTED：第 6 位，另一被跟踪标志。

char p_pri；优先数。最小值-100，最大值 127。其值越小，优先权越高。时钟中断程序每秒都要重算一次，供调度用。

char p_time；换入内存后的驻留时间或换到磁盘上的驻留时间(以秒为单位)。供对换用。

int p_addr；进程数据段起址，即 ppda 和 user 的起址。如数段在内存就是内存块号(每块 32 字)；如在磁盘上就是磁盘物理块号(每块 256 字)。

int p_size；数据段大小(以内存块为单位)。

int p_wchan；睡眠原因。

int*p_textp；指向 text 结构的指针。内含进程正文段的有关信息。

以上数据项在本部分内使用。其余项如下：

char p_sig；收到的软中断号码。

char p_uid；用户号。

char p_cpu；用于优先数计算。
char p_nice；用于优先数计算。
int p_ttyp；对应终端的 tty 结构地址。
int p_pid；进程号。
int p_ppid；父进程号。

UNIX 系统最多允许有 50 个进程，因此它保持一个 proc 表，其中每项是一个 proc 结构，相当于其它操作系统的 PCB（进程控制块）的主要部分。

系统中所有的正文段是通过一个正文表管理的，其中含有 40 项，每项是一 text 结构，含有对应正文段的描述与使用信息。

2．text 结构。它也是 text 数组中一个元素。其中有：

int x_daddr；正文段磁盘地址。
int x_caddr；正文段内存地址。
int x_size；正文段块数。
char x_count；访问计数。使用此正文段的进程数目。
char x_ccount；内存访问计数。使用此正文段内存中的进程数目。
int *x_iptr；含有此正文段文件的 i 节点指针。

这里 text 既是结构名，又是数组名。数组共 NTEXT（=40）项。

为了减少进程控制块对内存的占用量，UNIX 把其中最常用的部分，即该进程永远放在内存中的那一部分，作为 proc 结构常驻内存。而把另一部份，仅当进程正在运行时才用到的那部分 PCB 作为 user 结构与其它数据一起换入换出内存。

3．user 结构。形式上，它只有一个变量实例，u。其地址总是核心空间第 7 页地址，即 &u=140000（字节地址）。但实际上，由于地址 140000 是一虚拟地址，它可以映射到每个进程的内存 user 上，所以每个进程都有一个变量 u。user 结构的部分条目如下：

int u_rsav[2];
int u_qsav[2];
int u_ssav[2];
} 保留 r_5、r_6、用于正常或非正常返回。

int u_procp 指向本进程 proc 结构的指针；
int u_uisa[16]；16 字的数组。本进程用户空间地址寄存器样本；
int u_uisd[16]；16 字的数组。本进程用户空间说明寄存器样本；
int u_tsize；本进程正文段的块数；
int u_dsize；本进程数据区的块数；
int u_ssize；本进程栈区的块数；
char u_error；出错号码。

以上是与本部分讨论有关的项。其余项如下：

int u_fsav[25]；用于保留浮点寄存器。
char u_segflg；在文件系统中用于表示内存数据所在空间是用户还是核心空间。
char *u_base；用于文件传输，表示内存地址。
char *u_cnnut；用于文件传输，表示传送字节数。
char *u_offset[2]；用于文件传输，表示文件内相对位移量（字节）。
int *u_cdir；当前目录项 i 节点指针。
char u_dbuf[DIRSIZ]；保留当前用到的文件的路径名分量。DIRSIZ=14，表示分量名最大字符数。
char *u_dirp；指向当前目录文件名的指针。

struct {
  int u_ino；i 节点号。
  char u_name[DIRSIZ]；文件路径名分量。DIRSIZ=14。
} u_dent；存一个文件目录项。

char u_uid；有效用户号。
char u_gid；有效用户组号。

char u_ruid；实际用户号。

char u_rgid；实际用户组号。

int u_ofile[NOFILE]；用户打开文件表。NOFILE＝15，共15项。

int *u_pdir；父目录项i节点指针。

int u_signal[NSIG]；软件中断处理程序入口表。共20项，NSIG＝20。

int u_sep；指令(I)空间和数据(D)空间是否分离。

int u_arg[5]；当前系统调用中，用户传入的自变量。

int *u_aro；中断保留区中保留R0的栈内单元的地址。

int u_utime；本进程用户态运行时间。

int u_stime；本进程核心态运行时间。

int u_cutime[2]；子进程的用户态运行时间。

int u_cstime[2]；子进程的核心态运行时间。

int u_prof[4]；用于程序直方图计算。

char u_intflg；表示系统调用执行完成与否。

以上user结构中的35项，共占289字节处于ppda下部。其中u_rsav[2]固定为第一项不是任意的，其地址即user地址。由于user结构的实例是变量u，所以user的每个分量都必须加前缀u.来引用。例如u. u_rsav[1]指的就是当前运行进程的user结构中的分量u_rsav[1]。

上述数据结构之间的关系如图2.2所示。

2.3 地址映射管理

下面是与地址映射管理有关的两个程序及其框图。estabur根据进程映象各成分的大小及其它要求装配user结构中的样本寄存器，这与映象在内存中的地址无关。sureg则根据当前进程映象在内存中的地址装配硬件映射寄存器，最终实现用户虚拟空间到内存物理空间的映射。

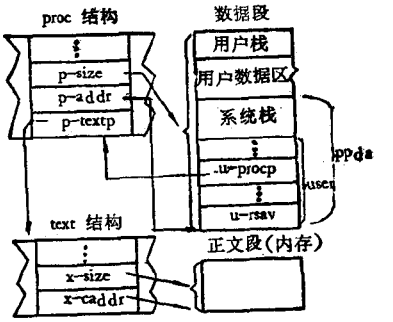


图 2.2 进程映象的结构

模块说明与框图

A、estabur(nt,nd,ns,sep) (1650—1730)

1．功能：在user结构中装配样本映射寄存器u_uisa和u_uisd以及硬件寄存器。

2．输入参数：

nt：正文段块数；

nd：数据区块数；

ns：栈区块数；

sep：指令(I)空间和数据(D)空间是否分离。

3．返回值：0，正常装配完成；

−1，参数超出允许值。

4．实现过程：(1) 检查参数，使其占用内存总量不超过每个进程允许最大块数。这个最大块数是在系统初启时由main程序确定的。

(2) 装正文段，从I空间第0页(u_uisa[0]，u_uisd[0])开始，相对物理块号从第0块开始

(3) 装数据段，如I、D空间分离则从第8页(u_uisa[8]、u_uisd[8])开始，否则从正文段所占的下页开始，相对物理块号从第16块开始，以便使用户空间跳过“进程数据区”(ppda)。

(4) 剩余页(如I、D不分离则到第7页)清0。

(5) 装栈段。从第15页(I、D分离)或第7页(I、D不分离)向下分配，相对物理地址从顶向下分配。如剩余不足一页，则页长度为最大页(128块)与它之差，扩展位ED置1表向下扩展，页起址为(上页起址−128)。

(6) 如ID空间不分离，则将已装配好的前8页样本寄存器复制到后8页上。

(7) 调用sureg按当前进程数据段、正文段起址装配硬件映射寄存器。

装配后的样本地址空间如图2.3所示（ID空间不分离）。

5．框图：图2.4。

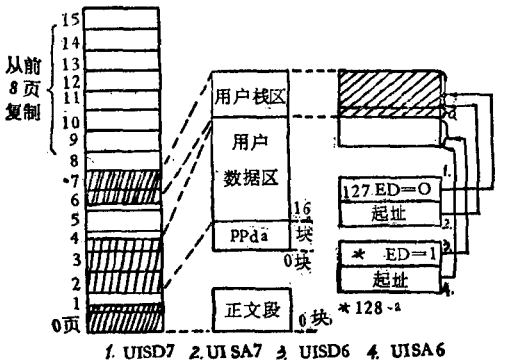


图 2.3 样本空间结构与用户栈的装配

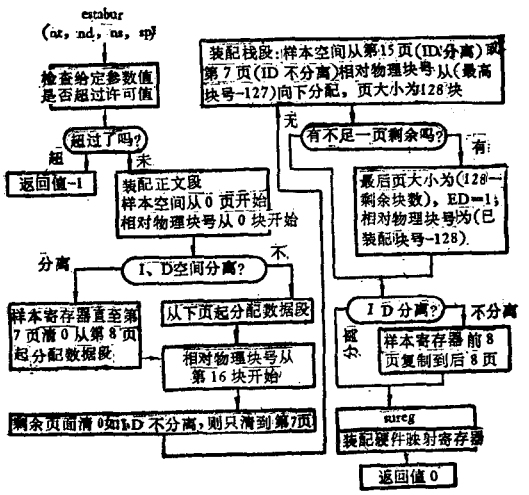


图 2.4 estabur

B. sureg (1739～1765)

1．功能：根据estabur在user结构中建立的样本寄存器内容和当前进程数据段，正文段的真实物理地址装配硬件映射寄存器。

2．实现过程：

(1) 查出本进程数据段的起始地址（u.u_procp→p_addr）。

(2) 将样本地址寄存器内容加上数据段起址装入硬件地址寄存器。

(3) 如有正文段，则从text数组对应表目查出正文段起址。

(4) 将样本说明寄存器内容复制到对应硬件说明寄存器中。如存取特征位是“非写”，则认为是正文段，将对应的地址寄存器内容改为以正文段为起址。

注：由于数据段的样本地址寄存器从第16块开始，所以数据段的第0页硬件地址寄存器指向的是内存的数据段第16块，因而底下16块作为“进程数据区”用户就不可存取了。

3．框图：图2.5。

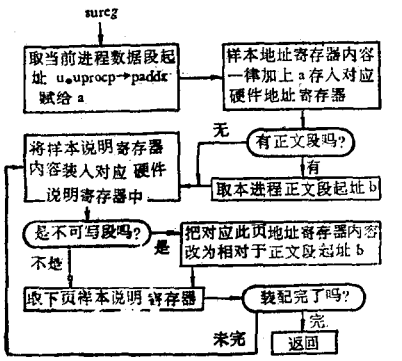


图 2.5 sureg

2.4 存储器的分配与释放

这里所说的存储器分配与释放指的是内存的用户空间和磁盘对换区部分。本模型系统只假定有一台单片两面的RK 05磁盘，共有4872块，其中前4000块用于文件系统。从4000到4871块为对换空间。内存可用空间是从UNIX核心所占的内存之外的一块（内存块大小为64字节，即32字）开始。

资源的分配与释放对于内存和磁盘对换区使用的程序是一样的，只是可用资源表不同。而且内存分配单位是块（32字），磁盘对换区分配单位是区（或称磁盘块，在不引起混淆的地方也简称为块）（256字）。释放单位也如此。内存与盘对换区可用空间总量由main程序调用mfree分别置以初值。

这里采用的算法叫做“最先满足算法”（first fit），即只要找到第一个满足需要的可用

段即可，不必考虑剩余多少。这种算法容许可用段在资源表中按地址增长方向排列，有利于释放后的可用段合并。

数据结构：

内存可用资源表和磁盘可用资源表均是50个 map 结构的数组。两个数组的起址分别是 coremap 和 swapmap。结构 map 的类型定义如下：

```
struct map
{char*m_size/* 指出可用段大小(块/区数)*/
char*m_addr;/*指出可用段起址(块/区号)*/}
```

每种数组内有效元素的排列是从低地址到高地址直至遇到第一个其 * m_size = 0 的 map 结构。每个 map 结构表示的可用段不能与其它可用段相连，否则应合并成一个 map 结构。

图 2.6 表示了可用资源表的形式。

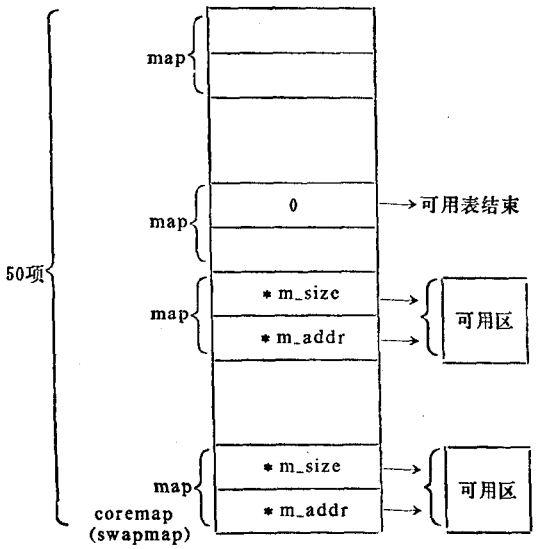


图 2.6 可用资源表格式

模块说明与框图：

A、malloc (mp,size) (2528～2547)

1. 功能：分配内存或盘对换区。

2. 输入参数：

mp—取值 coremap 或 swapmap，分别指向内存或盘对换区可用资源表起始入口（map 结构）；

size—申请资源单位数。内存单位是块(32 字)，盘对换区单位是区（256 字）。

3. 返回值：0，没有分配到；非零，分配到的存储段起始块（区）号。

4. 实现过程：从给定地址开始查可用资源表各个表目，找到第一个满足申请数目的表目。修改该表目，返回起始地址。如找不到合适表目，则返回 0 值。

5. 框图：图 2.7。

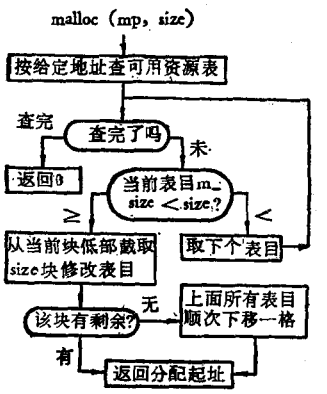


图 2.7 malloc

B、mfree (mp,size,aa) (2556～2588)

1. 功能：释放一段内存或盘对换区。

2. 输入参数：

mp—同 malloc 中含义；

size—释放资源单位数同 malloc 中含义；

aa—释放空间起始块号（内存）或区号（对换区）。

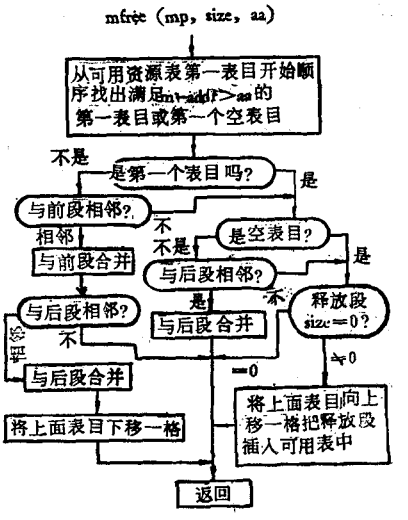


图 2.8 mfree

3．实现过程：按释放存储段的地址大小次序插入可用资源表对应表目中，如它与前面或后面可用存储段相邻则合并，修改相应表目或向上（向下）移动表目。

4．框图：图 2.8（见前页）。

2.5 控制台打印

系统利用下述程序以直接的，无缓冲的方式在操作员控制台终端上打印系统初置信息或硬件错误或系统紧急故障信息。因为这些程序不是中断驱动的，要动态等待前个字符传送完才能输出下个字符，所以不适于会话使用。

数据结构：

```
struct
{int rsr/* 终端接收机状态寄存器 */
 int rbr/* 终端接收机缓冲寄存器 */
 int xsr/* 终端发送机状态寄存器 */
 int xbr/* 终端发送机缓冲寄存器 */
}
```

这是一无名结构，表示控制台终端的四个外设寄存器类型。这组寄存器的起址是 KL=0177560（即 rsr 地址）。如要传送下个字符，xsr 的第 7 位必须不为 0。xbr 用于存放被传送字符。

```
char *panicstr;
```

保留最后调用 panic 时要打印的信息指针。用于控制台故障时检查。模块说明与框图：

A、panic (s) (2416～2424)

1．功能：在控制台打印信息。

2．输入参数：s—要打印的内容指针。

3．实现过程：

(1) 要打印信息内容送 panicstr。

(2) 调用 update（见第五部分）把块设备的专用块（superblock），i 节点（修改过）和延迟写的缓冲区写到盘上。

(3) 打印信息。格式是 panic：s（换行）

B、printf (fmt, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_a, x_b, x_c) (2340～2363)

1．功能：按规定格式打印参数值。

2．输入参数：

fmt：字符串，指出后面参数打印格式；

x_1
⋮
xc
} 要打印的参数。

3．实现过程：打印字符串 fmt 内%字符以前部分，再按%后面一个字符指出格式印出参数 x_1。继续重复上述过程，印出 x_2 以后各参数，直到 fmt 中字符读完。

C、printn (n, b) (2369～2376)

1．功能：把无符号整数 n 转换成以 b 为基底的无符号整数，以字符形式打印。

2．输入参数：

n—被转换的无符号整数；

b—作为基底的无符号整数。

D、putchar (c) (2386～2407)

1．功能：如果控制台开关 SW 打开且 xsr 第 7 位不为 0，则印出字符 c。

2．输入参数：

c—要打印字符。

E、deverror (bp, o_1, o_2) (2447～2455)

1．功能：在控制台上打印诊断信息。（只由 rkintr 调用，见 4.4 节）。

2．输入参数：bp—指向块设备缓冲首部的指针。o_1, o_2 为两个外设寄存器 RKER 和 RKDS 内容。

3．打印格式："err"on dev 主设备号/次设备号（换行），bn 缓冲区块号 er o_1 o_2

其中 o_1 和 o_2 为八进制数，其余为十进制数。

F、prdev (str, dev) (2433～2437)

1．功能：打印设备出错信息。

2．输入参数：

str—打印内容；

dev—设备名。

3．打印格式：str on dev 主设备号/次设备号。

其中 str 和 dev 为字符，其余为十进制数。

2.6 进程调度与交通控制

这一节介绍与进程调度，进程状态转换及进程通讯有关的几个程序。它们使用的公共数

据是上述的 proc 结构和 user 结构，即进程控制块。如前所述，UNIX 系统程序是以全局变量为中心的，proc 和 user 即属于这样的变量。因此，除了这里介绍的这几个程序外，系统中所有其它程序均可在任何地点，任何情况下，访问这些数据，从而改变自己或其它进程的状态。

swtch 是一低级调度程序，其任务是保留调用它的进程的现场；扫描 proc 表，找一合适的就绪进程运行。与一般操作系统的低级调度程序相比，它有三点值得注意：

1．swtch 运行期间不封锁中断。因而在它运行期间如果发生中断，则可能造成某些进程状态变化。于是可能产生这样的情况，系统中存在可以分配处理机的进程，而 swtch 却由于找不到这样的进程而使处理机处于等待状态，即整个系统处于等待状态。UNIX 的解决办法是，时钟每 20 ms 中断一次，使 swtch 摆脱等待状态重新扫描 proc 表，这时就可能找到可运行的进程了。而在一般的操作系统中，由于低级调度封锁中断，不会出现上述情况。

2．调用 swtch 如同调用其它任何子程序一样，不保留处理机状态字(PS)。因而调用进程的处理机状态(如优先级)可不知不觉地传入 swtch 选中的新进程。在UNIX 中，它的使用是很小心的。

3．在大多数操作系统中，低级调度是所有进程的转接站，因而不属哪一个进程。但在 UNIX 中，swtch 属于进程 0 的一部分。任一进程进入 swtch 以后首先恢复进程 0 的现场，然后进行调度工作，在它选中了某一进程后再恢复那个选中进程的现场。因此，进程 0 除了它的用于程序对换的主要代码 sched 之外，还包括有程序 swtch 中间的一段代码。

A、swtch(2178～2248)

1．功能：分配处理机。

2．返回值：永远是1。

3．实现过程：分为三个阶段，分属三个进程。

第一阶段：当前进程把自己环境指针 r_5和栈指针 r_6 保留在它的 u. u_rsav 中。

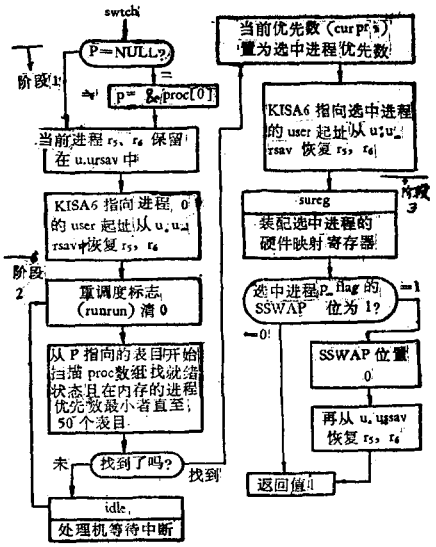


图 2.9 swtch

第二阶段：作为进程 0 的一部分运行，扫描 proc 表目，从当前进程的表目之后开始扫描 50 个表目，选出就绪状态且在内存而且优先数最低的进程运行。如果找不到这样的进程则调用汇编子程序 idle 使处理机处于等待状态，等待中断发生。

第三阶段：根据选中进程的 p_addr 装配核心空间第 6 页地址寄存器(KISA6)，进而找出该进程的u.u_rsav，从中恢复 r_5、r_6。然后装配选中进程的硬件映射寄存器。然而如果此进程的 p_flag 的 SSWAP 位为 1，则再从 u.u_ssav 恢复 r_5、r_6，以便作非正常返回(其用法见2.8 节 expand，newproc 程序)。

swtch 的返回值总是 1，只由 newproc 程序检查(见 2.8 节)。

注：swtch 程序用的变量 p 用于指示当前运行进程的 proc 表目地址，并指示下次扫描 50 个表目的起址。这是一静态局部变量，只能由 swtch 修改，但退出 swtch 时其值保留。这样有利于公平地选择进程运行。

4．框图：图 2.9。

程序 sleep 使当前进程放弃处理机，处于等待(SWAIT)或睡眠状态(SSLEEP)。其中有关状态修改的代码部分的处理机优先级提高为 6 级，以屏蔽时钟中断(6 级)以及其它外设中