第六章 进程结构

1、进程的状态和状态的转换

进程的基本状态可分为：

运行态　就绪态　睡眠态

进一步细分，又可分为九种状态：

①、进程在用户态下执行；

②、进程在核心态下执行；

③、进程已经准备好运行，在内存中就绪；

④、进程等待资源，在内存中睡眠；

⑤、进程处于就绪状态，因内存不足，被放在交换区上等待；

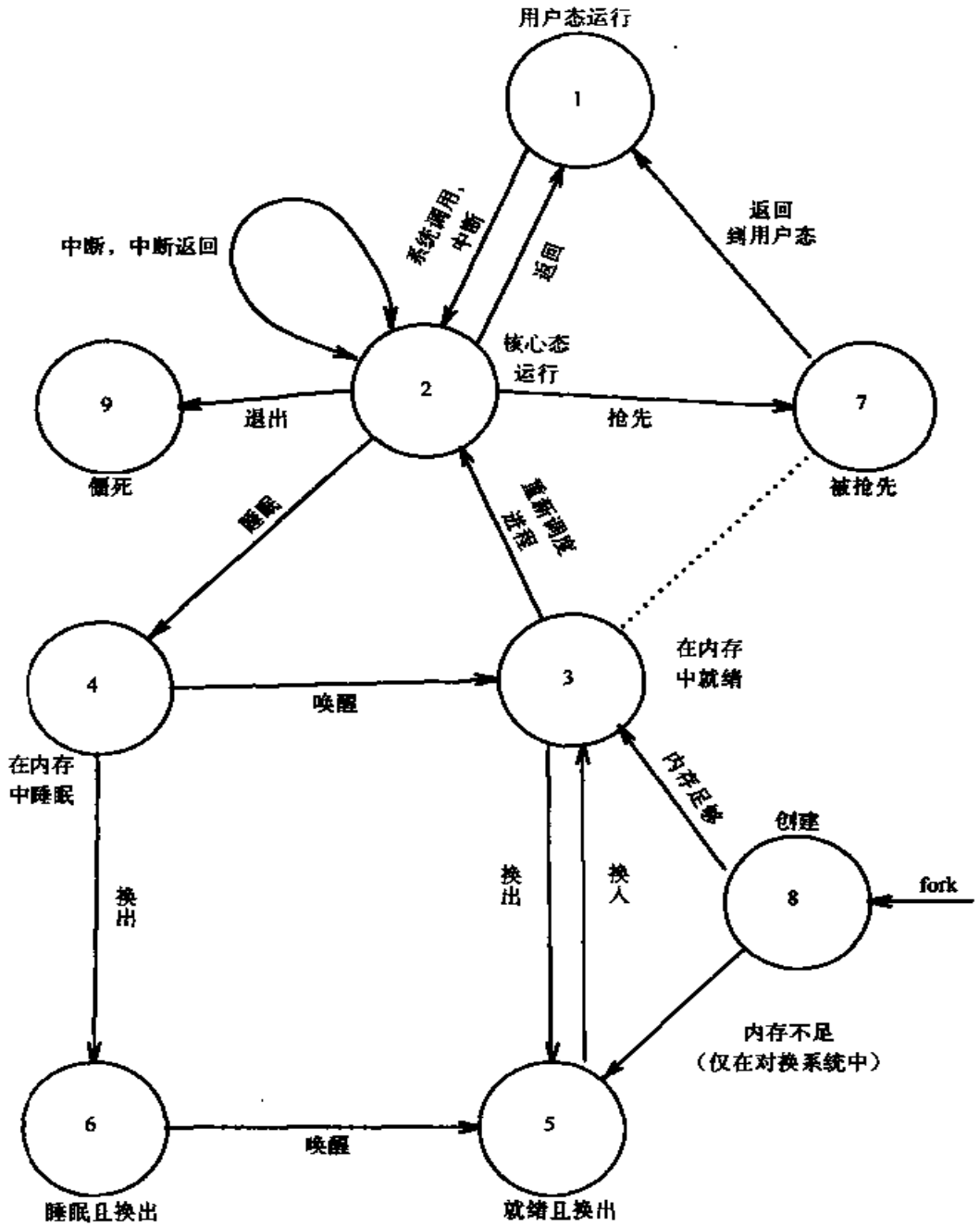
⑥、进程睡眠等待资源，因内存不足，被换到交换区上等待；

⑦、进程正从核心态返回用户态，但核心抢先于它做了上下文切换，以调度另外一个进程；

⑧、进程处于刚被创建的状态，此时进程既没有处于就绪状态，也没有进入睡眠状态；

⑨、进程执行了系统调用exit，处于僵死状态。此时进程刚消亡，并向父进程发送退出状态信息和计时统计信息。

进程在其生命周期中必然处在这九种状态之一，并且根据运行时间和条件的变化，在这九种状态之间进行转换，这种有方向规定的状态转换路径，就构成了进程的状态转换图——有向图。


用户态运行
1
中断，中断返回
系统调用，中断
返回
返回到用户态
核心态运行
2
9
退出
僵死
抢先
7
被抢先
睡眠
重新调度进程
4
唤醒
3
在内存中就绪
在内存中睡眠
换出
换出
换入
内存足够
创建
8
fork
内存不足
（仅在对换系统中）
6
唤醒
5
睡眠且换出
就绪且换出

有关进程抢先：

——任何进程都不能抢先另一个在核心中运行的进程

（1）、对于一个在核心态下运行的进程，如果没有因等待资源而睡眠，或者被中断，或者执行完毕准备退出，则它永远占用处理机而不会发生上下文切换（调度其他进程）。

为什么？

有关进程抢先：

——任何进程都不能抢先另一个在核心中运行的进程

（2）、对于一个在用户态下运行的进程，也不会直接发生抢先。

如果该用户态下的进程运行时间较长，则在时钟中断处理程序（核心态）运行完毕、准备返回到用户态时发生抢先。

进程状态3与状态7的区别：

状态3 —— 在内存中就绪

通常（除创建状态外）进程是因执行系统调用（申请资源）进入睡眠后，又被唤醒后进入就绪状态. 因此进程被重新调度后就进入状态2（核心态下）继续运行。

状态7 —— 被抢先

正在运行的进程从核心态正要返回到用户态时，因运行时间足够长了，发生进程切换，从而进入就绪状态。当重新被调度后，进入用户态继续执行。

直接效果 ——

一个是进入核心态运行；一个是进入用户态运行

2、进程的特征：

①、每个进程在核心进程表（**proc**数组）都占有一项，在其中记录了进程的状态信息；
②、每个进程都有一个“每进程数据区（**per process data area —— ppda**）”，保留相应进程更多的信息和核心栈；
③、处理机的全部工作就是在某个时候执行某个进程；
④、一个进程可生成或消灭另一进程；
⑤、一个进程中可申请并占有资源；
⑥、一个进程只能沿着一个特定的指令序列运行，不会跳转到另一个进程的指令序列中去，也不能访问别的进程的数据和堆栈。（抗病毒传播的重要原因之一）

3、进程的解释

在UNIX系统中进程的概念包含什么意义？

在较高级的方面

进程是一个重要的组织概念。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，可以相互合作和竞争，而真正活动的部件如处理机和外部设备则是看不见的。

在较低级方面

进程是不活动的实体，而处理机则是活动的，处理机的任务就是对进程进行操作，处理机在各个进程映像之间转换。

4、进程映像

进程映像虽然包括很多方面，但关键问题是存储器映像。当一个进程暂时退出处理机时，它的处理机映像（也就是各个寄存器的值）就成为存贮器映像中的一部分。当该进程再次被调度使用处理机时，又从存储器映像中恢复处理机映像。

一般说来，进程的结构，即进程映像是指其存储器映像，由下列几部分组成：

① 进程控制块（PCB）
② 进程执行的程序，即共享正文段（TEXT）
③ 进程执行时所需要使用的数据，即数据段（DATA）
④ 进程执行时使用的工作区，即栈段

1）、进程控制块 PCB

PCB包括两部分信息：

一部分是不论进程当前是否在处理机上运行，系统都要查询和修改的一些控制信息，它们构成一个数据结构**proc**（进程基本控制块）。所有进程的**proc**结构就构成了核心“进程表”——“进程表”中每一项都是一个存放某进程控制信息的**proc**结构。

另一部分信息当进程不在处理机上运行时，系统不会对它们进行查询和处理，这些信息构成另一个数据结构**user**，它是对**proc**的扩充，称为“进程扩充控制块**user**结构”。

进程基本控制块 PROC

```
struct  proc {
        进程状态标志域
        指向U区的指针
        进程属主 UID
        进程标识数 PID
        睡眠地址
        进程优先级等调度参数
        软中断信号域
        进程运行计时域，用于计算进程的优先级
}
```

进程扩充控制块 USER

```
struct  user  {
        指向proc的指针
        真正用户标识号和有效用户标识号
        计时器域，用于统计汇总
        指向软中断信号处理函数的指针
        控制终端
        错误标识域
        系统调用返回值
        用于读写的I/O参数
        进程的当前目录和当前根
        用户文件描述符表（用户打开文件表）
        其它标识域
}
```

2）、共享正文段：

进程执行的程序用可再入码编写并可使若干个进程共享执行的部分构成共享正文段（纯正文段，不可修改段），它包括再入的程序和常数。

3）、数据段：

进程执行时用到的数据构成数据段。如果进程执行的某些程序为非共享程序，则它们也构成数据段的一部分。

4）、栈段：

进程在核心态下运行时的工作区为核心栈，在用户态下运行时的工作区为用户栈。在进行函数调用时，栈用于传递参数，保护现场，存放返回地址以及为局部动态变量提供存储区。

核心栈的结构和操作方式与用户栈相同，只是存放进程在核心态下调用函数时的有关信息。

5、进程映像在存贮器中的分布：

存放进程映像的地方有两个：

①、内存

②、磁盘交换区——内存的扩充部分

进程的映像分为常驻内存部分和非常驻内存部分

常驻内存部分：

进程基本控制块**proc**

进程共享正文段的控制信息**text**

非常驻内存部分：

进程扩充控制块**user**

进程在核心态下的工作区（核心栈）

数据段

进程在用户态下的工作区（用户栈）

共享正文段

处理机正在执行本进程时：

该进程映像全部（或所需部分）在内存中

处理机执行其它进程时：

该进程映像所占用的内存可能被分配给其它进程，如果是这样，则当前进程的映像要被调出到对换区中去，等到重新满足运行条件并获得处理机时，再次被调入内存运行。

UNIX系统中所采用的进程映像调入调出措施，可以大大提高内存的利用率，增加系统的吞吐量，以便同时为尽可能多的分时用户提供服务。

在进程映像占用的内存被分配给其他进程之前，不但该进程的程序和数据需要调出内存，该进程的控制信息也被调出内存。但为了该进程能够再次被调入内存，内存中需要保留一部分必要的信息，这就把进程控制信息也分成了常驻内存和非常驻内存两部分：

常驻内存控制信息块

是系统需要经常查询以及恢复整个进程映象时所不可缺少的信息。

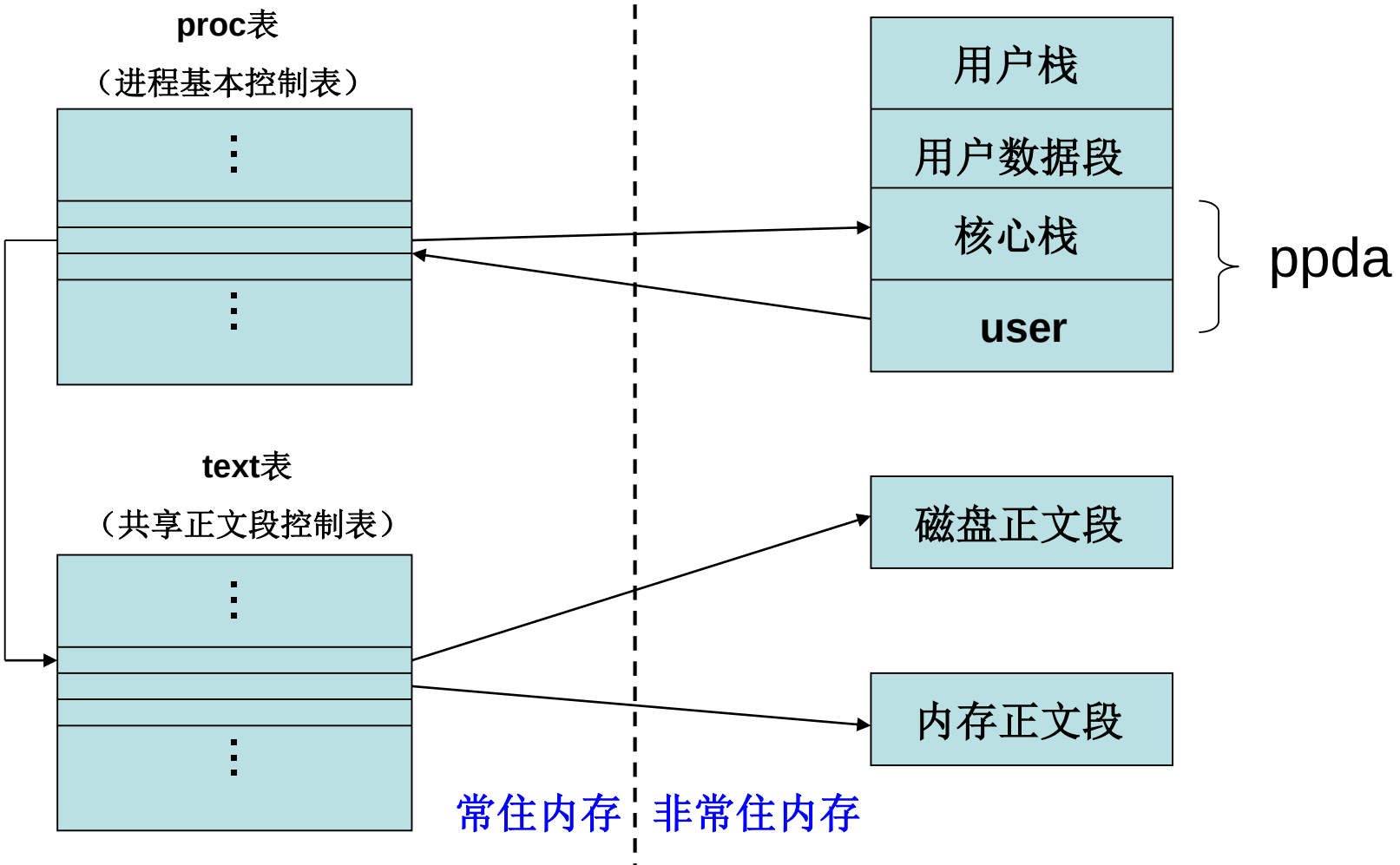
非常驻内存控制信息块

可以随进程状态的变化而在内外存之间交换的进程控制信息中的其余部分。

为了方便进程映像在内外之间交换，UNIX系统中把进程非常驻内存部分作为一个整体，占用连续的存贮区，其顺序是：首先是user结构（进程扩充控制块）和核心栈，然后是数据段和用户栈。

进程user结构和核心栈合并构成进程的“本进程数据区——ppda区（per process data area）。

ppda区以及用户数据段和用户栈，要调入内存则一起调入，要调出则一起调出。图示为进程映像的组织方式。



6、进程上下文（context）

一个进程的上下文包括五个方面：

①、被进程正文所定义的进程状态
②、进程所使用的全局变量和数据结构的值
③、机器寄存器的值
④、进程表项proc结构和user结构中的值
⑤、用户堆栈和核心堆栈中的值

“执行一个进程”——指系统在该进程的上下文中执行，也就是进程的上下文确定和限制了进程的运行环境和空间。

核心从一个进程转到另一个进程执行时，叫做“上下文切换”，也就是系统从一个进程上下文确定的环境换到另一个进程上下文确定的环境中去执行。并且保留前一个进程的必要信息，能够在以后需要时又再切换回前一个进程，并恢复它的执行。

相关说明：

①、用户态和核心态之间的转换是执行状态的转换而不是上下文的切换。

②、在遇到中断信号时，核心在当前进程的上下文中对中断服务，而不论该中断是否是本进程引起的。系统在核心态下对中断进行服务，而不是产生或调度一个特殊进程来处理中断。被中断的进程无论是在用户态下执行还是在核心态下执行，核心都保留必要的信息以便处理中断完后能恢复本进程的执行。

③、进程映像和进程上下文的区别

进程映像——主要是指进程的结构和内部组织形式

进程上下文——构成进程映像的各部分的各种取值的集合

7、进程状态及映像的对应关系

①、运行状态

此时进程正在占用处理机，进程的全部映像（或正在运行的部分）驻留在内存中，存储管理器（MMU）中装配的是本进程映象的地址映射参数，系统变量U（即U区）与本进程的user结构对应。

处理机可能是在执行用户进程本身，也可能是在执行系统程序，前一种情况称为进程在用户态下运行，后一种情况称为进程在核心态下运行。

在核心态下的进程虽然执行的是操作系统内部的程序，但由于所用的系统栈和进程控制块仍然是本进程的，所以此时进程可能是在调用系统调用，或是在进行中断处理，信号处理等。

②、就绪状态

处于就绪状态下的进程基本具备了运行条件，正在等待使用处理机。这时进程的映像可能全部在内存，也可能只有需要运行的部分在内存。**MMU**装配的不是本进程映像的地址映射参数，系统变量**U**也不与本进程的**user**结构对应。

系统中通常有若干的不同优先级的就绪队列，每个就绪队列中有若干的就绪进程在排队等待运行。

③、睡眠状态

进程不具备运行条件，需等待某种事件的发生，无法继续执行下去。此时进程的映像根据内存的空闲程度，可能全部在内存中、可能部分在内存中、也可能全部都在交换区上。

在程序实现上，造成进程睡眠的唯一直接原因是调用了**sleep()**函数，调用**sleep()**的原因有多种：

a、进程在运行过程中要使用某种资源，如缓冲区，但由于该资源已被占用，不能立即得到满足，不得不进入睡眠状态，等待其它进程释放该资源。

b、进程实施同步或互斥，如父进程对子进程进行跟踪时，子进程等待父进程发控制命令，或父进程等待子进程完成某一次操作等。

c、等待输入*/*输出操作的完成

d、进程完成某一项任务后，暂时停止自身的运行，等待新任务的来临，如**0**号进程或**shell**进程等。

8、睡眠与唤醒

一般而言，一个进程除非自己调用了**sleep**函数，否则是在执行一个系统调用期间进入睡眠的（**why?**）：

该进程执行一个操作系统陷入（**trap**），进入核心，然后可能进入睡眠等待某个资源（或软资源或硬资源）。

进程在一个事件上睡眠是它们处于睡眠状态，直到该事件发生。

当进程等待的事件发生时，等待该资源的进程就被唤醒并进入“就绪”状态，而不是直接进入“运行”状态。

sleep函数运行时完成下面三项主要工作：

①、将进程的状态标志设置为睡眠

②、记录下睡眠原因

③、修改进程优先级。这个优先级是进程下次被唤醒时所具有的竞争处理机的能力，它因不同的睡眠原因而异，因为不同的事件要求响应的紧急程度是不同的。

系统中各种睡眠原因相应的优先权

睡眠原因	睡眠优先数
进程对换	PSWP
inode操作	PINOD
块设备操作	PRIBIO
	PRIUBA
	PIERO
管道操作	PPIPE
虚拟文件操作	PVFS
等待操作	PWAIT
上锁操作	PLOCK
资源等待	PSLEP
用户状态	PUSER
初始值	NZERO

优先数小，优先级高

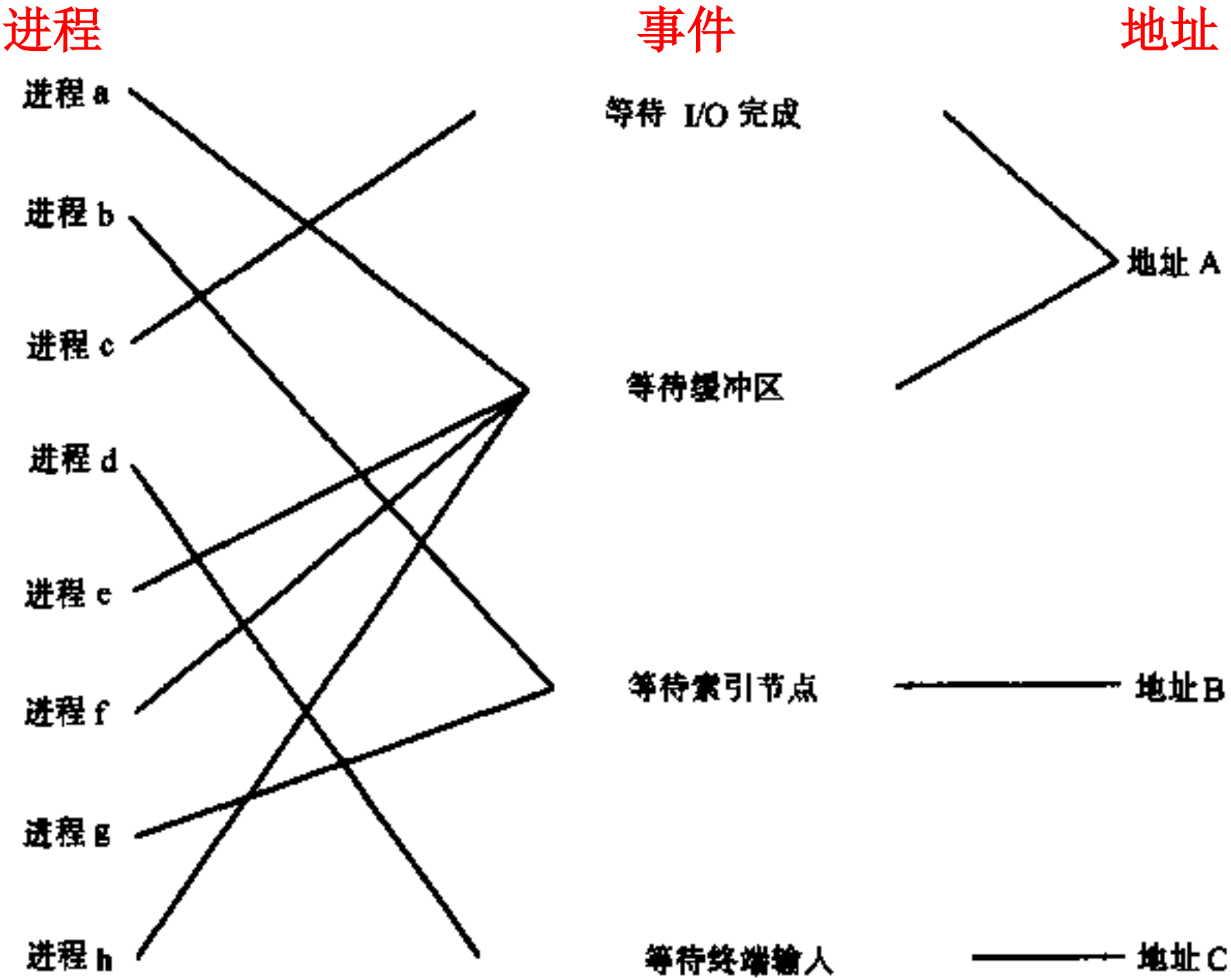
优先数大，优先级低

睡眠事件及其地址：

“进程在一个事件上睡眠”或“进程睡眠等待一个资源”，就是等待处理该事件（或操作该资源）的程序代码段映射到本进程核心（因为是系统调用）地址空间中。

例如读写磁盘操作，当A进程正占用磁盘——执行磁盘驱动程序，设置磁道、扇区、读写数量和数据传输模式等参数，初始化磁盘缓冲区等操作时——B进程就不能再映射（执行）磁盘驱动程序的代码段，否则将破坏A进程设置的参数。

A进程设置完参数后，将睡眠等待（同步）或不等待（异步）物理磁盘的操作，磁盘操作完毕时，发出中断信号，由当前正在运行的程序执行磁盘中断处理程序，通知A进程接收数据，唤醒包括B进程在内的所有等待磁盘操作的进程。



睡眠在事件上的进程及映射到地址上的事件

9、中断处理

中断的分类：

- 硬件中断——来自时钟和各种外部设备
- 可编程中断——来自“软件中断”指令
- 例外中断——中断的特例，来自页面错误

核心处理中断的操作顺序：

① 保护现场 —— 保存当前进程的上下文

② 确定中断源 —— 根据中断向量查找中断处理程序

③ 调用中断处理程序 —— 完成处理任务

④ 中断返回 —— 恢复被中断程序的上下文

中断处理程序的算法

算法　**inthand**

输入：无

输出：无

{

保存当前上下文，并创建一个新的上下文层；

确定中断源，得到中断信号号；

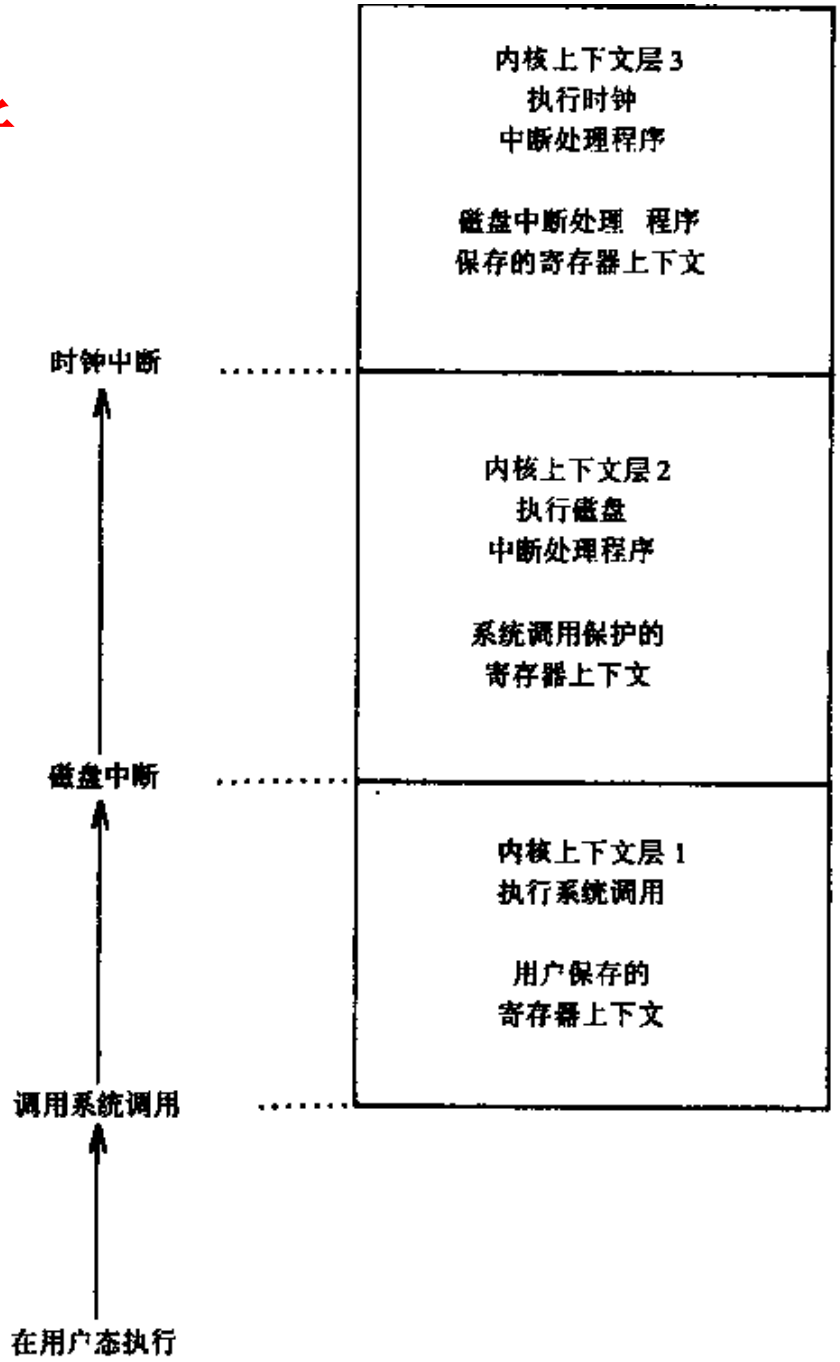
查找中断向量，找到中断处理程序入口；

调用中断处理程序，完成具体任务；

恢复前一个上下文，继续执行被中断程序；

}

中断的例子



每一层都
可能包含
多个栈帧

10、系统调用接口

在为UNIX系统编写的C语言编译程序中，定义了一个与系统调用名字一一对应的函数库，使得应用程序能够执行系统调用。

库函数中包含一条操作系统陷入（trap）指令，把程序的运行模式由用户态转变为核心态。

每一个库函数在执行trap指令时，使用与机器有关的方式向核心传送一个操作系统陷入号，即系统调用号。

C编译程序中的库函数是在用户态下运行的，当库函数执行trap指令时，产生一个中断信号，调用中断处理程序处理中断请求（进入核心态，执行系统调用），中断向量就是系统调用号。

系统调用接口是中断处理程序的特例！

```
算法 syscall  /* 系统调用的算法 */
输入：系统调用号
输出：系统调用的结果
{
    在系统调用表中找出对应于系统调用号的表项；
    确定系统调用的参数数目；
    将参数从用户地址空间拷贝到u区；
    为废弃返回面保存当前上下文(见第6.4.4节)；
    调用内核中的系统调用代码；
    if(在系统调用的执行中有错)
    {
        将用户保存的寄存器上下文中的寄存器0设置为错误号；
        将用户保存的寄存器上下文中的PS寄存器的进位位打开；
    }
    else
    将在用户保存的寄存器上下文中的寄存器0,1设置为系统调用返回值；
}
```

数据寄存器

状态寄存器

系统调用实例分析：

```
char  name[ ] = “file”;
main()
{
        int  fd;
        fd = creat(name, 0666);
}
```

更一般的用法：

```
char name[ ] = “file”;
main()
{
        int  fd;
        if ((fd = creat(name,0666)) < 0)
            printf(错误信息);
}
```

```
# code for main
58:  mov    &0x1b6, (%sp)          # 将0666压到堆栈上
5e:  mov    &0x204, -(%sp)         # 把变量name压到堆栈上
64:  jsr      0x7a                 # 调用creat的C语言库程序
# library code for creat
7a:  movq  &0x8,%d0                # 将creat的陷入号8移到数据寄存器0
7c:  trap     &0x0                 # 操作系统陷入，调用8号中断处理程序
7e:  bcc     &0x6  <86>            # 如果进位位为零（正确），则转到地址86
80:  jmp     0x13c                 # 否则（错误），转到地址13c
86:  rts                           # 从子程序creat返回到main函数
# library code for errors in system call
13c: mov   %d0,&0x20e              # 把寄存器0中的错误号移到20e单元（变量errno）
142: movq &-0x1,%d0                # 将常数-1放到数据寄存器0中，置错误标志
144: mova %d0,%a0
146: rts                           # 从子程序creat返回到main函数
```