第三章 Operating system's processes 进程管理

本章主要内容

1. 进程的概念
2. 进程的描述
3. 进程状态及其转换
4. 进程控制
5. 进程互斥
6. 进程同步
7. 进程通信
8. 死锁问题
9. 线程的概念
10. 线程的分类与执行

3.1 进程的概念

3.1.1 程序的并发执行

程序：算法支配下的计算机指令与数据集合（算法 + 指令 + 数据）。

单道程序系统的程序的顺序执行：在算法逻辑的引导下依次执行指令的过程。

模块内顺序执行，

模块间的顺序执行，

程序间的顺序执行。

顺序执行的特点：**顺序性，封闭性，可再现性**

多道程序系统中，程序执行环境发生随机变化

新任务（程序）的进入和进程的启动，旧进程的完成退出和任务（程序）的结束，使得内存中的并发进程数处在无法预知的随机动态变化中。

多道程序执行的特点：交叉性，开放性，不可再现性。

3.1 进程的概念

4. 程序的并发 (concurrent) 执行

所谓程序的**并发执行，是指两个或两个以上程序同处于开始和结束之间的中间状态。**能够参与并发执行的程序称为并发程序。

自中断引进之后，计算机系统中的程序在执行中往往会被中断，这就是说内存中的进程们是走走停停，停停走走，你停我走，你走我停，形成若干进程交替执行的局面，我们说这些交替执行的进程是在并发执行。所以，**中断是并发程序实现的必备前提！**

3.1 进程的概念

- 程序并发性

所谓程序并发是指在计算机系统中同时存在多个程序。宏观上看来，这些程序是同时向前推进的。程序的并发性体现在两方面：1）用户程序与用户程序之间并发执行；2）用户程序与操作系统程序之间并发执行。

- 并发与并行

并发与并行是两个不同的概念。程序并行执行要求微观上的同时，即在绝对的同一时刻有多个程序同时向前推进。这要求机内有多个 CPU ，每个 CPU 上有一个进程（程序），我们说这若干个 CPU 上的进程是并行的。

程序并发执行不要求微观上的同时，让 CPU 分时地、轮番地运行内存里的每道程序。外部看来，每道程序都有一个 CPU（虚 CPU ）、（似乎）都在同时向前推进，其实机内只有一个 CPU 。

3.1 进程的概念

并发，即多个程序已“起程出发”，各自“走走停停”，获得了CPU就“走”，被剥夺了CPU就“停”。

并行，是多个程序从“出发”一直“走”到“结束”，中间从不“停顿”。这样看来，并行是并发的一个特例。

- 程序的并发执行特点：若干个逻辑上相互独立的程序或程序段同时在系统中执行，这些程序的执行在时间上是重叠的，一个程序的执行尚未结束，另一个程序的执行已经开始。

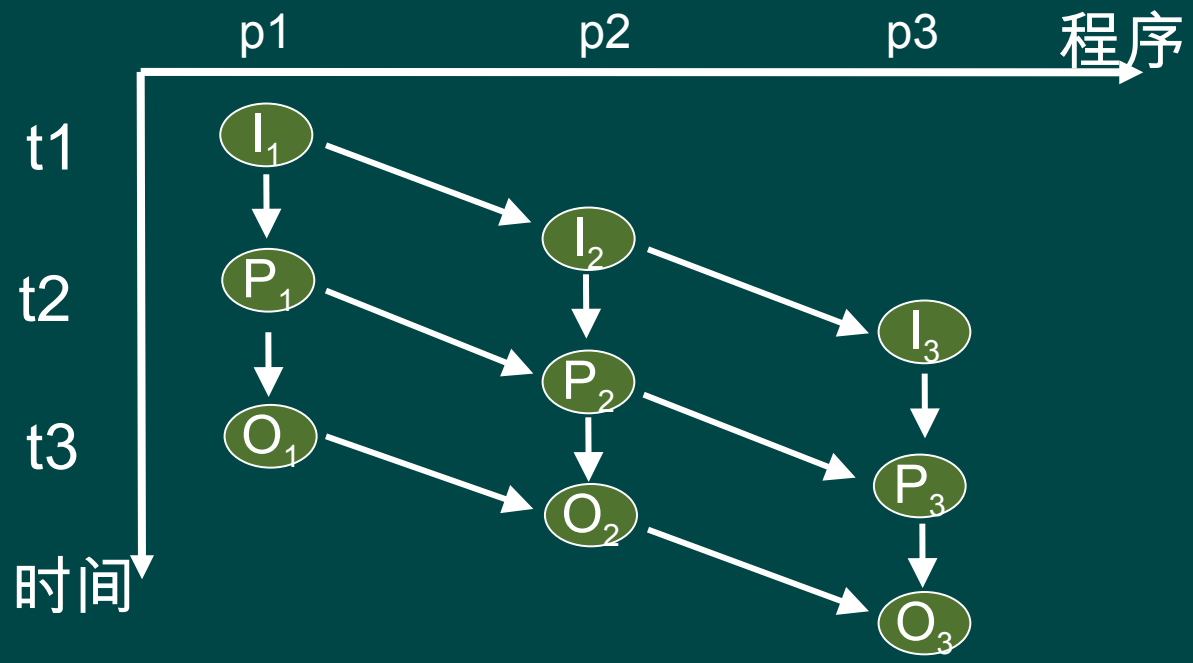
宏观上：在一个时间段中几个进程同时处于活动状态。

微观上：任一时刻仅有一个进程在处理器上运行。

3.1 进程的概念

3.1 进程的概念

- 并发程序设计举例
  - **某程序需要循环执行输入、计算、输出三个过程。因此可以设计为三个程序段I（ input ）、P（ process ）、O（ output ）**



3.1 进程的概念

- 程序的并发执行带来的影响
  - **程序的并发执行导致资源共享与资源的竞争，使得程序执行过程不可避免的失去了封闭性和可再现性。**
  - **程序与计算不再一一对应。**
  - **程序并发执行的相互制约。**

3.1 进程的概念

3.1.2 进程的定义

这里简单的定义为，**进程是一个正在参与并发执行的程序。它由程序、数据、进程控制块（ PCB ）共同组成。** PCB 记录了进程停顿时所有的现场信息，作为下次继续运行的依据。进程是可以并发执行的计算部分。

其它定义：

- 进程是一个独立的可以调度的活动
- 进程是一个抽象实体，需要分配和释放资源
- 程序在处理机上执行时的活动称为进程
- 一个进程是一系列逐一执行的操作

3.1 进程的概念

进程的执行

进程的执行是进程正在 CPU 上运行，正在占有 CPU。程序的执行是程序调入内存，安排进入进程状态。只要程序的进程没有完结，就说该程序正在执行，不管其进程是否正在占有 CPU。

进程与程序的联系与区别：

- 进程是一个动态概念，而程序是一个静态概念。
- 进程具有并发特征，而程序没有。
- 进程是竞争计算机系统资源的单位。
- 不同的进程可以对应的是同一个程序。

3.1 进程的概念

进程特征

- 动态性：进程是程序的执行；
- 并发性：多个进程可同存于内存中，能在一段时间内同时运行；
- 独立性：独立运行的基本单位，独立获得资源和调度的基本单位；
- 异步性：各进程按各自独立不可预知的速度向前推进；
- 结构特征：由程序段、数据段、进程控制块三部分(PCB)组成。

3.2 进程的描述—PCB

- 为了描述一个进程和其它进程以及系统资源的关系和刻画一个进程在各个不同时期所处的状态，人们采用了一个与进程相联系的数据块，称为进程控制块（ PCB ）
- 系统利用 PCB 来控制和管理进程，所以 PCB 是系统感知进程存在的唯一标志 .
- 进程与 PCB 是一一对应的 (一个进程有且只有一个 PCB).
- PCB 内容
  - 描述信息
    - 进程标识符 (process ID)
    - 用户标识符 (user ID)
    - 家族关系

3.2 进程的描述—PCB

- 控制信息
  - 当前状态
  - 优先级 (priority)
  - 代码执行入口地址
  - 程序的外存地址
  - 运行统计信息（执行时间、页面调度）
  - 进程间同步和通信；阻塞原因
- 资源管理信息
  - 虚拟地址空间的现状
  - 打开文件列表

3.2 进程的描述—PCB

CPU 现场保护结构

- 寄存器值（通用、程序计数器 PC 、状态 PSW ，地址包括栈指针）
- 指向赋予该进程的段 / 页表的指针。

- 进程上下文

进程上下文包含：

每个进程执行过的、执行时的以及待执行的指令和数据；

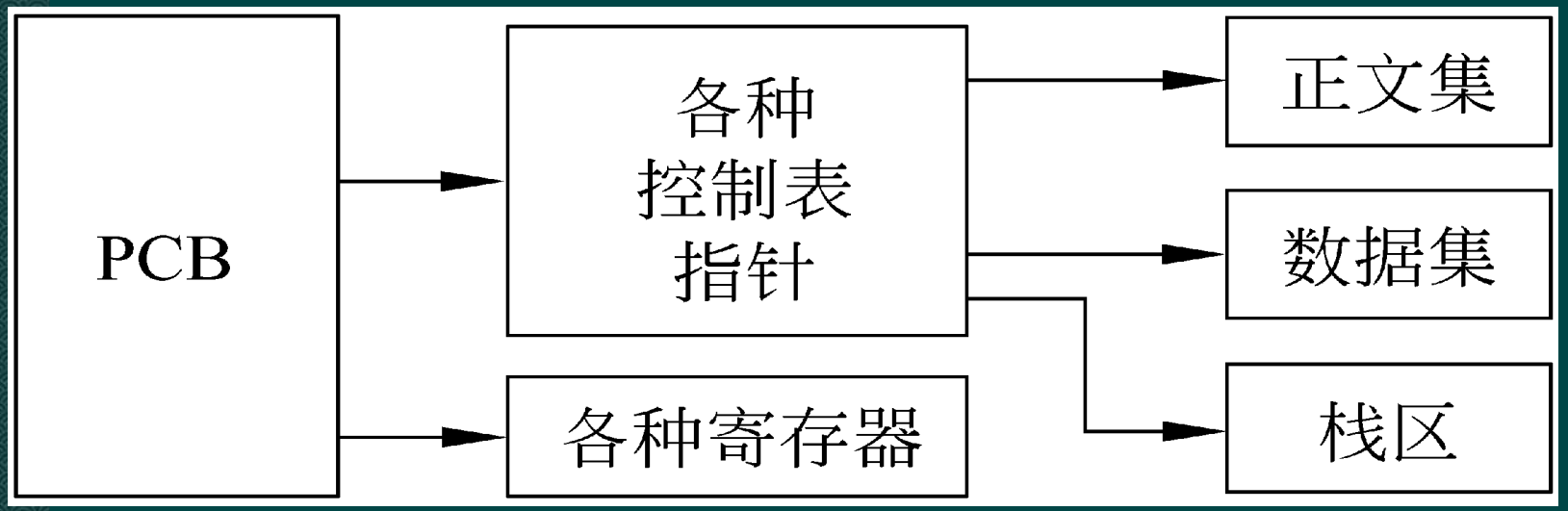
在指令寄存器、堆栈、状态字寄存器等中的内容。

- 进程上下文切换

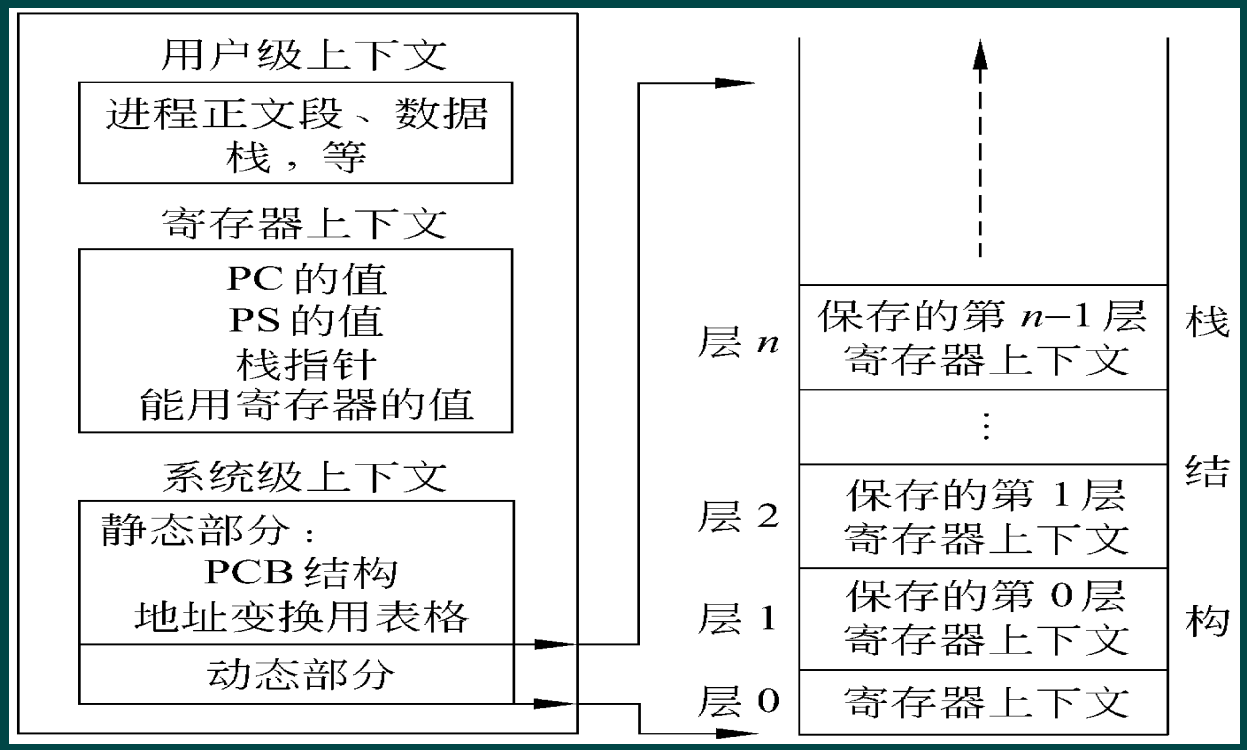
3.2 进程的描述—PCB


进程 P1 执行
中断或进程调用
保存进程 P1 正文至 PCB1
系统进程执行
选取新进程 P2，恢复 P2 上下文
新进程 P2 执行

3.2 进程的描述—PCB



3.2 进程的描述—PCB


用户级上下文
进程正文段、数据
栈，等
寄存器上下文
PC 的值
PS 的值
栈指针
能用寄存器的值
系统级上下文
静态部分：
PCB 结构
地址变换用表格
动态部分
层 n
层 2
层 1
层 0
保存的第 n–1 层
寄存器上下文
保存的第 1 层
寄存器上下文
保存的第 0 层
寄存器上下文
寄存器上下文
栈
结
构

3.2 进程的描述—PCB

- 进程空间大小
  - 进程空间大小与处理机的位数有关
- 空间的划分
  - 系统空间：存放操作系统的程序、信息、以及重要的大部分进程信息。
  - 用户空间：存放用户的进程文本和相关信息。

3.3 进程状态及其转换

进程有三种基本状态：

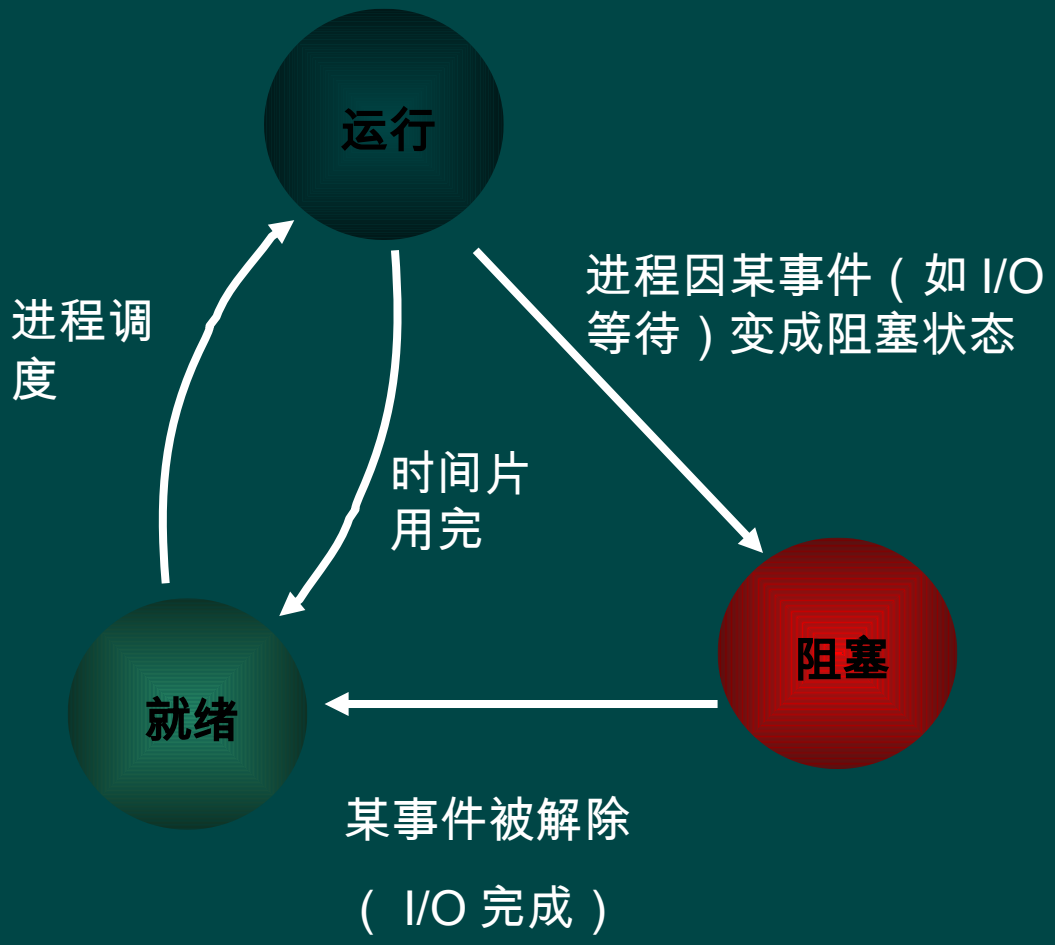
就绪状态（ Ready ）：存在于处理机调度队列中的那些进程，它们已经准备就绪，一旦得到 CPU ，就立即可以运行。这些进程所处的状态为就绪状态。

运行状态（ Running ）：正在运行的进程所处的状态为运行状态。

等待状态（ Wait / Blocked ）：若一进程正在等待某一事件发生（如等待输入输出工作完成），这时即使给它 CPU ，它也无法运行，称该进程处于等待状态、阻塞、 睡眠、封锁状态。

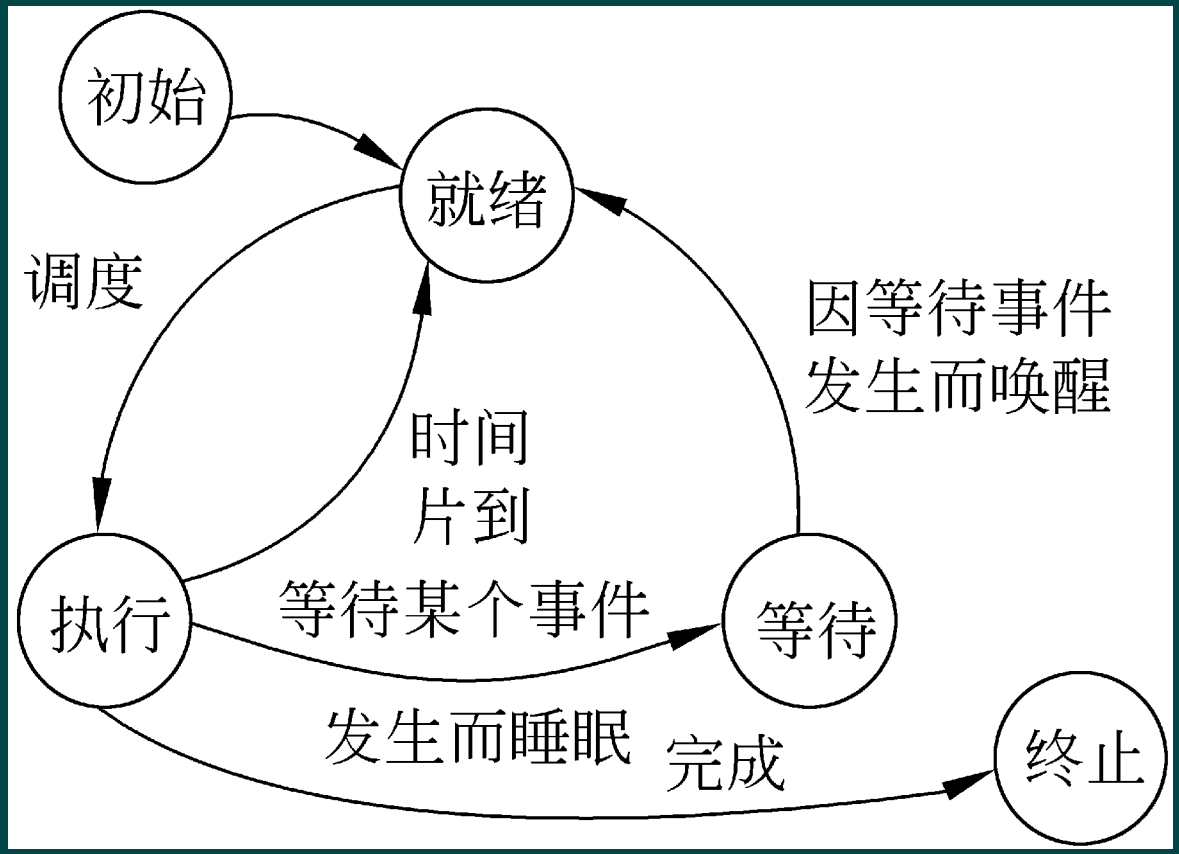
进程在生命消亡前处于且仅处于三种基本状态之一。

3.3 进程状态及其转换



3.3 进程状态及其转换

五状态进程模型



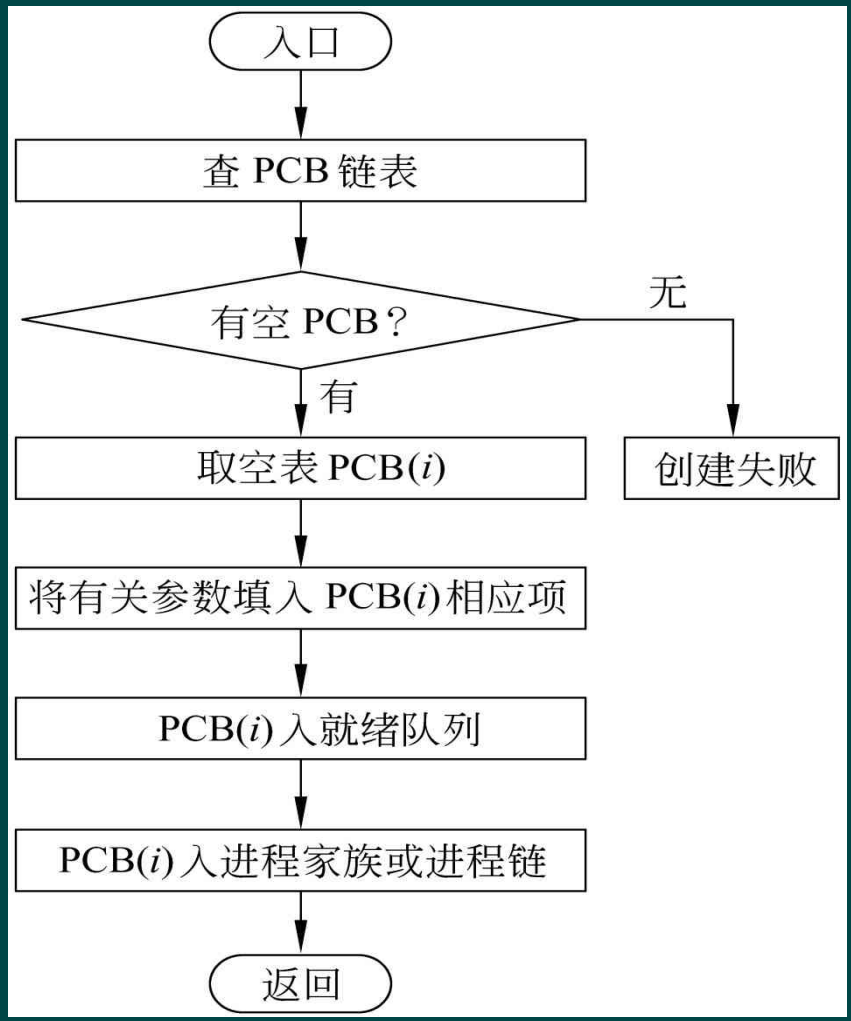

HEWLETT PACKARD
LaserJet

3.4 进程控制

- 定义：系统使用一些**具有特定功能的程序段**来创建、撤销进程以及完成进程各状态间的转换，从而达到多进程高效率并发执行和协调、实现资源共享的目的。进程控制一般由**原语**来实现。
- 原语：执行时不可中断的指令单位。
  - **机器指令级**
  - **功能级**

3.4 进程控制

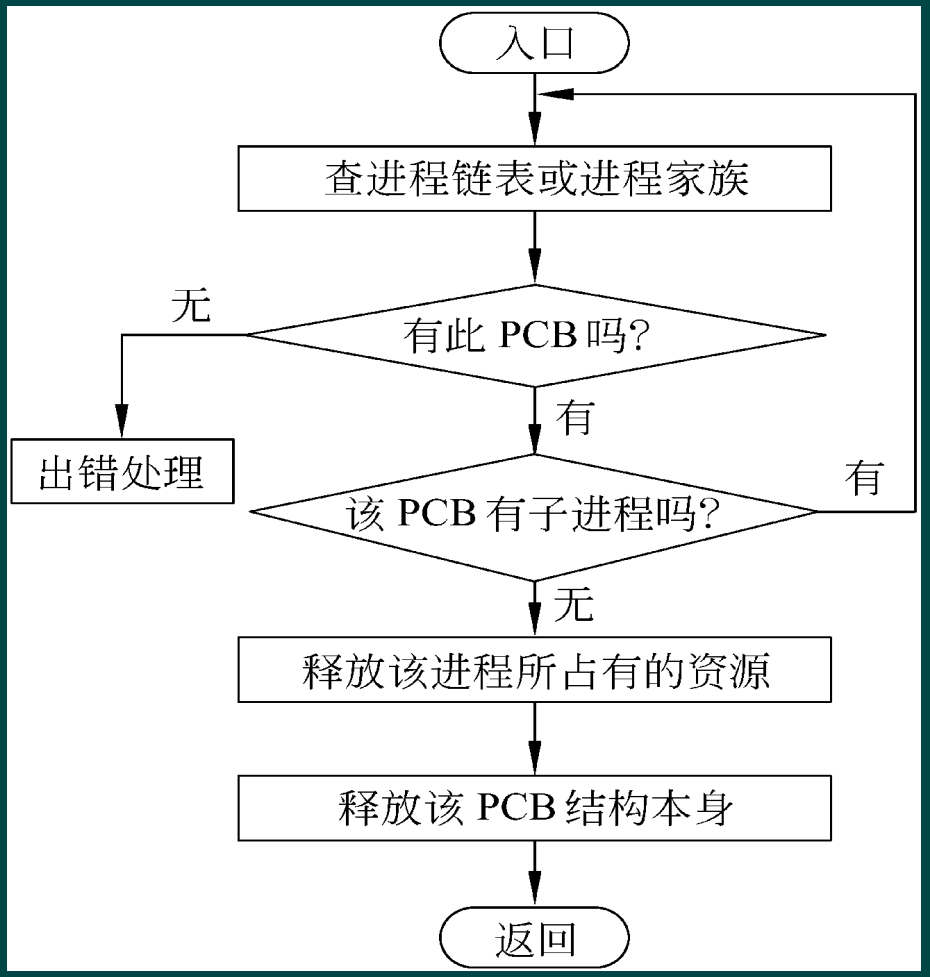
- 进程创建
  - 系统程序模块统一创建
  - 由父进程创建



3.4 进程控制

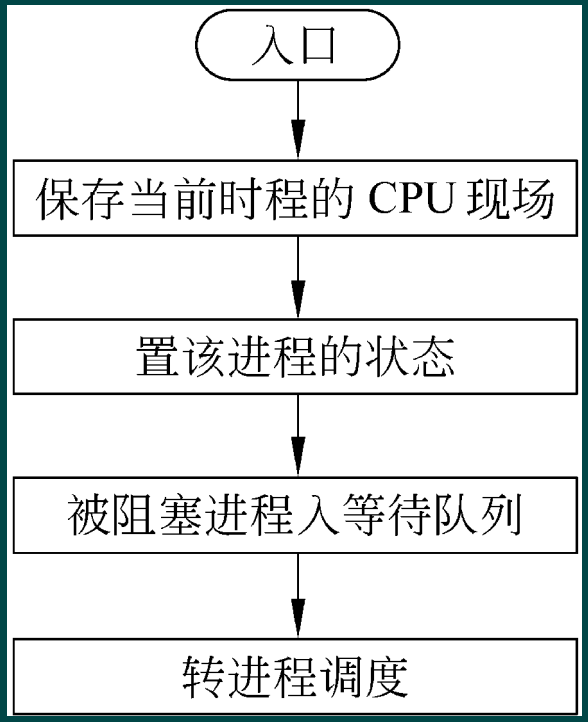
进程撤销

- 进程已经完成所要求的任务
- 由于某种错误导致非正常终止
- 祖先进程要求撤销某个子进程

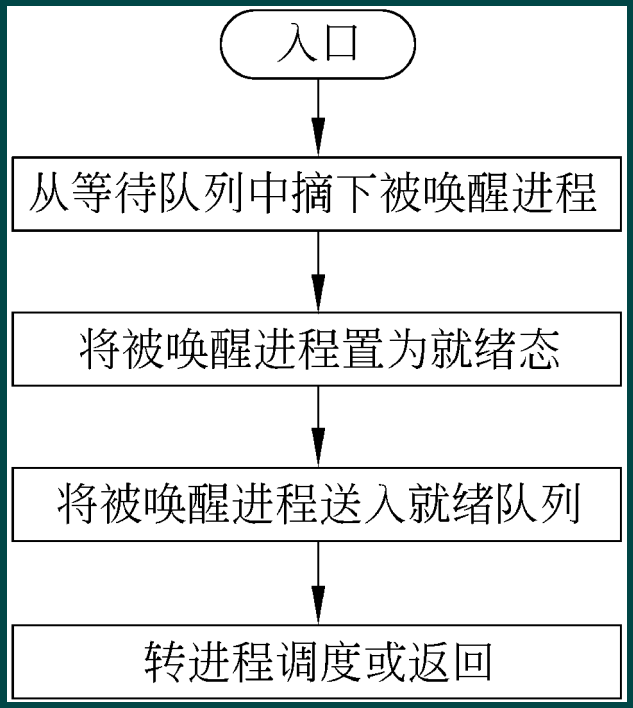


3.4 进程控制

- 阻塞
  - 当一个进程所期待的某一事件尚未出现时，**该进程调用阻塞原语将自己阻塞。**进程阻塞是进程的自身的一种主动行为。



- 唤醒
  - 处于阻塞状态的进程是绝不可能叫醒它自己的，它必须由它的**合作进程用唤醒原语唤醒它。**




2005.11.11

3.5 进程互斥

3.5.1 基本概念

3.5.2 互斥的加锁实现

3.5.3 信号量和 P ，V 原语

3.5.4 用 P ，V 原语实现进程互斥

3.5 进程互斥（基本概念）

- **临界资源（ critical resource ）：一次仅允许一个进程访问的资源。**
  - 如：进程A，B共享一台打印机，若让它们交替使用则得到的结果肯定不是我们希望的。
  - 临界资源可能是硬件，也可能是软件：变量，数据，表格，队列等。
  - 并发进程对临界资源的访问必须作某种限制，否则就可能出与时间有关的错误，如：联网售票。

3.5 进程互斥（基本概念）

- 临界区

  把不允许多个并发进程交叉执行的一段程序称为临界区（ critical region ）

  临界区是由属于不同并发进程的程序段共享某个公用数据或公用变量而引起的。

  临界区是对某一临界资源而言的，对于不同临界资源的临界区，它们之间不存在互斥。

- 互斥

  一组并发进程中的一个或多个程序段，因共享某一公有资源而导致他们必须以一个不允许交叉执行的单位执行。即不允许两个以上的共享该资源的并发进程同时进入临界区称为互斥。

3.5 进程互斥（基本概念）

- 互斥准则
  - 并发进程享有平等的、独立的竞争共有资源的权利
  - 不在临界区的进程不能阻止其他进程进入临界区
  - 每次最多只有一个进程进入临界区
  - 每个进程能在有限的时间内进入临界区

3.5 进程互斥（锁）

- 对临界区加锁实现互斥
  - 当某个进程进入临界区之后，它将锁上临界区，直到退出临界区。
  - 并发进程进入临界区之前，测试该临界区是否上锁。
- 加锁后临界区程序描述如下：

```
lock(key[S])
< 临界区 >
unlock(key[S])
```

3.5 进程互斥（锁）

设 key[S]=1 为临界区可用，key[S]=0 表示临界区不可以用，那么 unlock(key[S]) 一条语句就可以实现：

key[S]←1

对于 lock(key[S]) 而言，简单方法：

```
Begin local v
        repeat
        v←key[s]
    Until v=1
        key[s]←0
end
```

3.5 进程互斥（信号量）

- 信号量
  - 锁机制存在弊端（多个进程竞争导致同时进入）
  - 信号是交通中常用的设备，通过信号颜色变化来实现交通管理
  - 在操作系统中，信号量是一个整数，大于等于零表示可使用的资源实体数，小于零表示正在等待使用临界区的进程数

3.5 进程互斥

- 1965 年，由荷兰学者 Dijkstra 提出（所以 P 、 V 分别是荷兰语的 test (proberen) 和 increment (verhogen) ）
- 信号量的数值仅能由 P 、 V 原语来改变。
- P 、 V 原语与锁的差异在于进程无法进入临界区时进入等待队列，而不是循环测试。

3.5 进程互斥

- P 原语操作的主要动作是：
  - sem 减 1 ；
  - 若 sem 减 1 后仍大于或等于零，则进程继续执行；
  - 若 sem 减 1 后小于零，则该进程被阻塞后进入与该信号相对应的队列中，然后转进程调度。

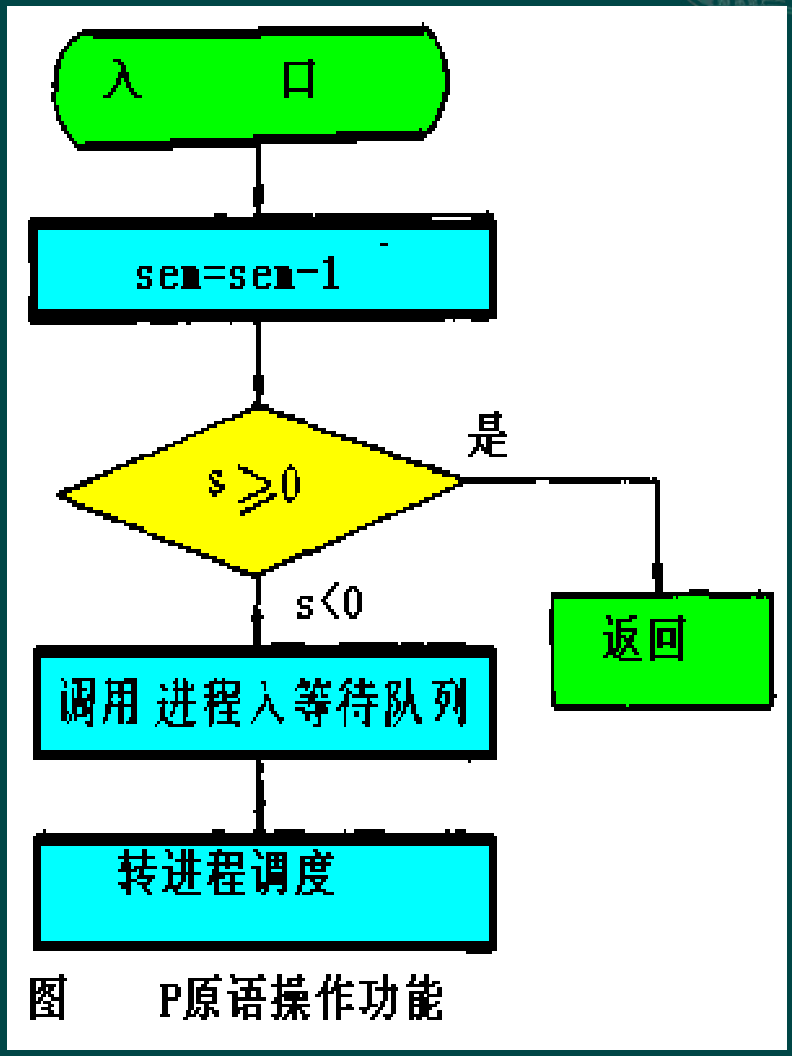


图　P原语操作功能

3.5 进程互斥

- V 原语操作的主要动作是：
  - sem 加 1 ；
  - 若相加结果大于零，则进程继续执行；
  - 若相加结果小于或等于零，则从该信号的等待队列中唤醒一等待进程，然后再返回原进程继续执行或转进程调度。

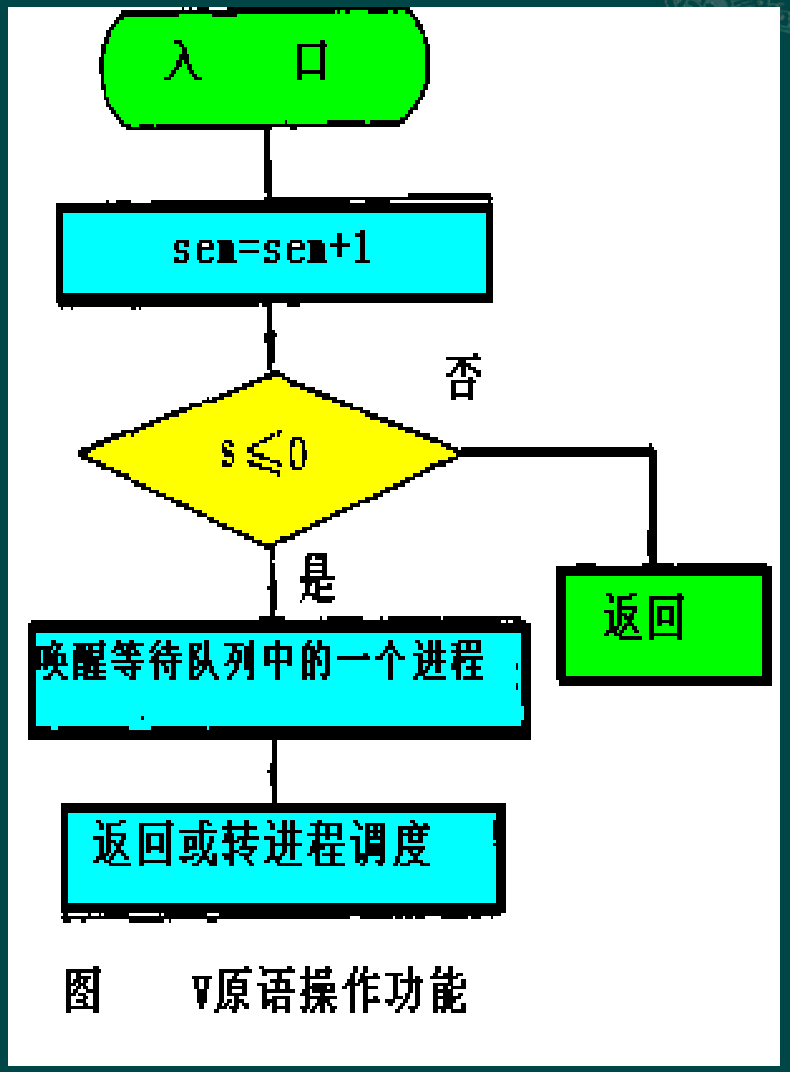


图　　V原语操作功能

3.5 进程互斥

◈ P、V 原语实现

```
P(sem) :
        begin
          封锁中断
          val[sem] = val[sem]-1
          if val[sem] < 0
          保护当前进程 CPU 现场
          当前进程状态置为“等待”
          将当前进程插入信号 sem 等
待队列
          转进程调度
          fi
          开放中断
        end
```

```
V(sem):
        begin
          封锁中断
          val[sem] = val[sem] + 1
          if val[sem]<=0
            local k
            从 sem 等待队列中选取一
等待进程，将其指针置入 k 中
            将 k 插入就绪队列
            进程状态置“就绪”
          fi
          开放中断
        end
```

3.5 进程互斥

- 利用 P ，V 原语和信号量解决并发进程的互斥问题。

用信号量实现两并发进程 Pa ， Pb 互斥的描述如下：

设 sem 为互斥信号量，其取值范围为（ 1 ， 0 ， -1 ）。其中 sem=1 表示无进程进入名为 S 的临界区， sem=0 表示进程 Pa 或 Pb 已进入名为 S 的临界区， sem=-1 表示进程 Pa 和 Pb 中，一个进入临界区，而另一个等待进入临界区。

```
Pa ：
        P （ sem ）
        <S>
        V(sem)::
        :
        :
```

```
Pb ：
        P(sem)
        <S>
        V(sem):
        :
        :
```

3.6 进程同步

- 同步的概念
  - 异步环境下的**一组并发进程因直接制约（如互相发送消息等）而进行的互相合作、互相等待，使得各进程按一定的速度推进的过程称为进程间的同步**。具有同步关系的一组并发进程称为**合作进程**，合作进程间互相发送的信号称为**消息或事件**。
  - 可用过程

    wait（消息名）

    表示进程等待某合作进程发来的消息，

    用过程

    signal（消息名）

    表示向某合作进程发送消息。

3.6 进程同步

- 私用信号量

  用 wait(消息名) 与 signal(消息名) 的方式是实现进程同步的一种方法 (见课本 p59-60) 。

  使用信号量的方法也可实现进程间的同步。

  把各进程之间发送的消息作为信号量看待。这里的信号量只与制约进程及被制约进程有关而不是与整组并发进程有关。该信号量为**私用信号量** (Private Semaphore) 。

  与私用信号量相对应，我们称**互斥时**使用的信号量为**公用信号量** (Public Semaphore) 。

3.6 进程同步

- 用 P ，V 原语操作实现同步
  - 步骤：
    - 为各并发进程**设置私用信号量**
    - 为私用信号量**赋初值**
    - 利用 p 、 v 原语和私用信号量**规定各进程的执行顺序**

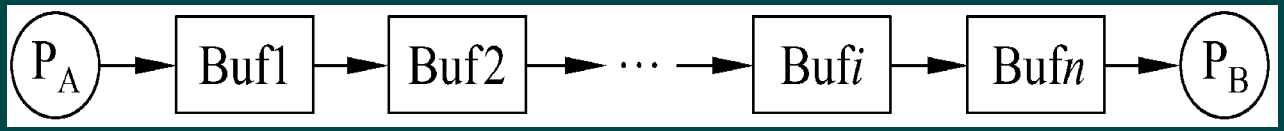
3.6 进程同步

- 用 P ， V 原语操作实现同步
  - 实例：
    - 设进程 PA 和 PB 通过缓冲区队列传递数据。 PA 为发送进程、 PB 为接收进程。发送过程 deposit(data) ，接收过程 remove(data) 。且数据的发 / 接收过程满足如下条件：

      1) 在 PA 至少送一块数据入一个缓冲区之前 ， PB 不可能从缓冲区中取出数据；

      2)PA 往缓冲队列发送数据时，至少有一个缓冲区是空的 ；

      3) 由 PA 发送的数据块在缓冲队列中按先进先出 (FIFO) 方式排列。



描述发送过程 deposit(data) 和接收过程 remove(data)

。

3.6 进程同步

1）设 Bufempty 为进程 PA 的私用信号量，Buffull 为进程 PB 的私用信号量；

2）令 Bufempty 的初始值为 n(n 为缓冲队列的缓冲区个数），Buffull 的初始值为 0；

3）描述：

PA：deposit(data)：

```
begin local x
        P(Bufempty);

        按 FIFO 方式选择一个空缓冲区 Buf(x);

        Buf(x)←data

        Buf(x) 置满标记

        V(Buffull)
end
```

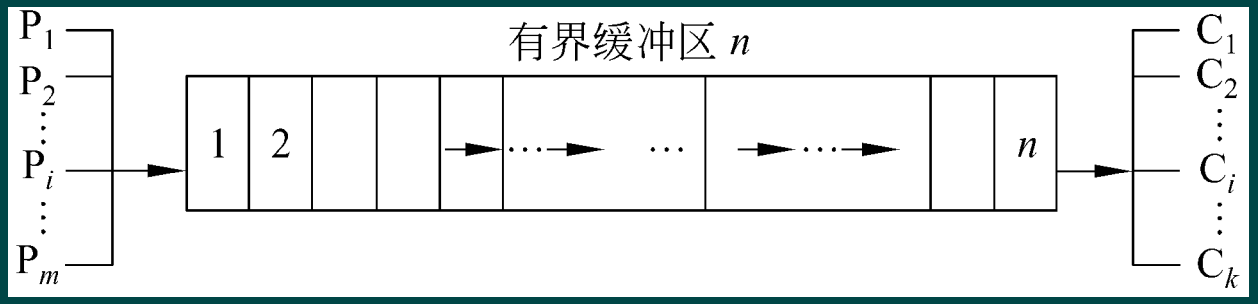
PB：remove(data)：

```
begin local x
        P(Buffull);
        按 FIFO 方式选择一个装满数据的缓冲区 Buf(x)
        data←Buf(x)
        Buf(x) 置空标记
        V(Bufempty)
end
```

3.6 进程同步

- 生产者 - 消费者问题
  - 计算机系统中的每个进程都可以消费（使用）或生产（释放）某类资源。这些资源可以是硬件也可以是软件资源。
  - 当某一进程使用某一资源时，该进程为消费者。当某一进程释放某一资源时，它就相当于生产者。

例子：若干进程通过有限的共享缓冲区交换数据。其中，“生产者”进程不断写入，而“消费者”进程不断读出；共享缓冲区共有 N 个；任何时刻只能有一个进程可对共享缓冲区进行操作。



3.6 进程同步

（1）消费者想接收数据时，有界缓冲区中**至少有一个单元是满的**。设信号量 full 表示缓冲区中有数据单元，初值为 0

（2）生产者想发送数据时，有界缓冲区中**至少有一个单元是空的**。设信号量 avail 表示缓冲区中空单元数，初值为 n

（3）有界缓冲区是临界资源，因此，各个生产者，消费者进程之间都必须互斥。信号量 **mutex 表示可用缓冲区个数**，初值为 1

```
deposit(data):
    begin
        P(avail)
        P(mutex)
        送数据入缓冲区某单元
        V(full)
        V(mutex)
    end
```

```
remove(data):
    begin
        P(full)
        P(mutex)
        取缓冲区中某单元数据
        V(avail)
        V(mutex)
    End
```

3.7 进程通信

- 通信（ communication ）：信息的传递
- **进程间通信：**
  - 低级通信：**只能传递状态和整数值（控制信息），如进程互斥和同步所采用的信号量。速度快，但是传送信息量小 。**
  - 高级通信：**能够传送任意数量的数据 。目的在于交换信息。**

3.7 进程通信

- 进程的通信方式（一）
  - **主从式** (master-servant system) 通信
    - **主进程可自由地使用从进程的资源或数据；**
    - **从进程的动作受主进程的控制；**
    - **主从关系是固定的。**

    如：终端控制进程和终端进程。

3.7 进程通信

- 进程的通信方式（二）
  - 会话式 (dialogue system)
    - 通信通信进程双方可分别称为**客户进程和服务进程**。其中，客户进程调用服务进程提供的服务。
    - **客户**进程在使用服务进程所提供的服务之前，必须得到**服务进程的许可**。
    - 服务进程根据**客户**进程的要求提供服务，但对所提供服务的控制由服务进程自身完成。
    - **客户**进程和服务进程在进行通信时**有固定连接关系**。


2005.06.09

3.7 进程通信

- 进程的通信方式（三）
  - **消息或邮箱机制**
    - 无论接收进程是否已准备好接收消息，发送进程都将把所**要发送的消息**送入缓冲区或邮箱。
    - 只要存在**空缓冲区**或邮箱，发送进程就可以发送消息。
    - 与会话系统不同的是发送进程与接收进程之间**无直接连接**关系。
    - 发送进程与接收进程之间存在有用来存放被传送消息的**缓冲区或邮箱**。

图 消息的组成

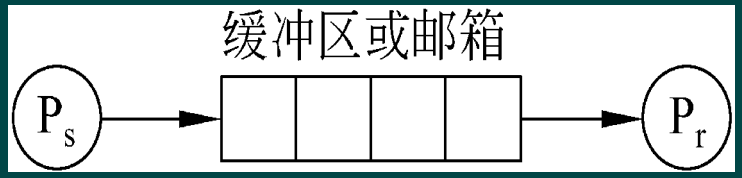


图 缓冲区或邮箱结构

3.7 进程通信

- 进程的通信方式（四）
  - **共享存储区方式**
    - 共享存储区方式不要求数据移动。**两个需要互相交换信息的进程通过对同一共享数据区 (shared memory) 的操作来达到互相通信的目的。**这个共享数据区是每个互相通信进程的一个组成部分。

3.7 进程通信

- 消息缓冲机制
  - **消息缓冲机制中所使用的缓冲区为公用缓冲区，使用消息缓冲机制传送数据时，两通信进程必须满足如下条件：**
    - 某进程把消息**写入缓冲区**时，禁止其它进程对该缓冲区消息队列访问(**互斥**)。否则，将引起消息队列的混乱。接收进程正从消息队列中**取消息**缓冲时，也应禁止其它进程对该队列的访问。
    - 当缓冲区中无消息存在时，**接收**进程不能接收到任何消息。

3.7 进程通信

- 消息缓冲机制

公用信号量 mutex 为缓冲区互斥信号量，初值为 1 。 SM 为接收进程私用信号量，表示等待接收的消息个数，初值为 0 。发送进程调用 send(m) 将消息 m 送往缓冲区，接收进程调用 receive(m) 将消息 m 从缓冲区读往自己的数据区 .send(m) 和 receive(n) 分别描述为：

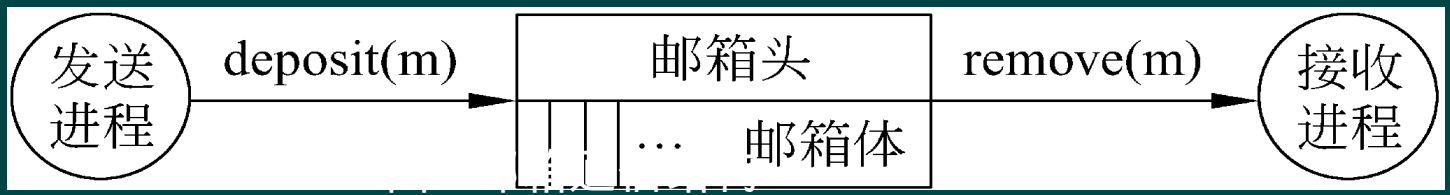
```
send(m):
        begin
         向系统申请一个消息缓冲区
         P(mutex)
         将发送区消息 m 送入新申请的
消息缓冲区
        把消息缓冲区挂入接收进程的稍
息队列
         V(mutex)
        V(SM)
        end
```

```
receive(n):
       begin
          P(SM)
          P(mutex)
           摘下消息队列中的消息 n
           将消息 n 从缓冲区复制到
接收区
          释放缓冲区
           V(mutex)
          End
```

3.7 进程通信

邮箱通信

- **邮箱通信就是由发送进程申请建立一与接收进程链接的邮箱。发送进程把消息送往邮箱，接收进程从邮箱中取出消息，从而完成进程间信息交换。**
- **邮箱由邮箱头和邮箱体组成。其中邮箱头描述邮箱名称、邮箱大小、邮箱方向以及拥有该邮箱的进程名等。邮箱体主要用来存放消息。**

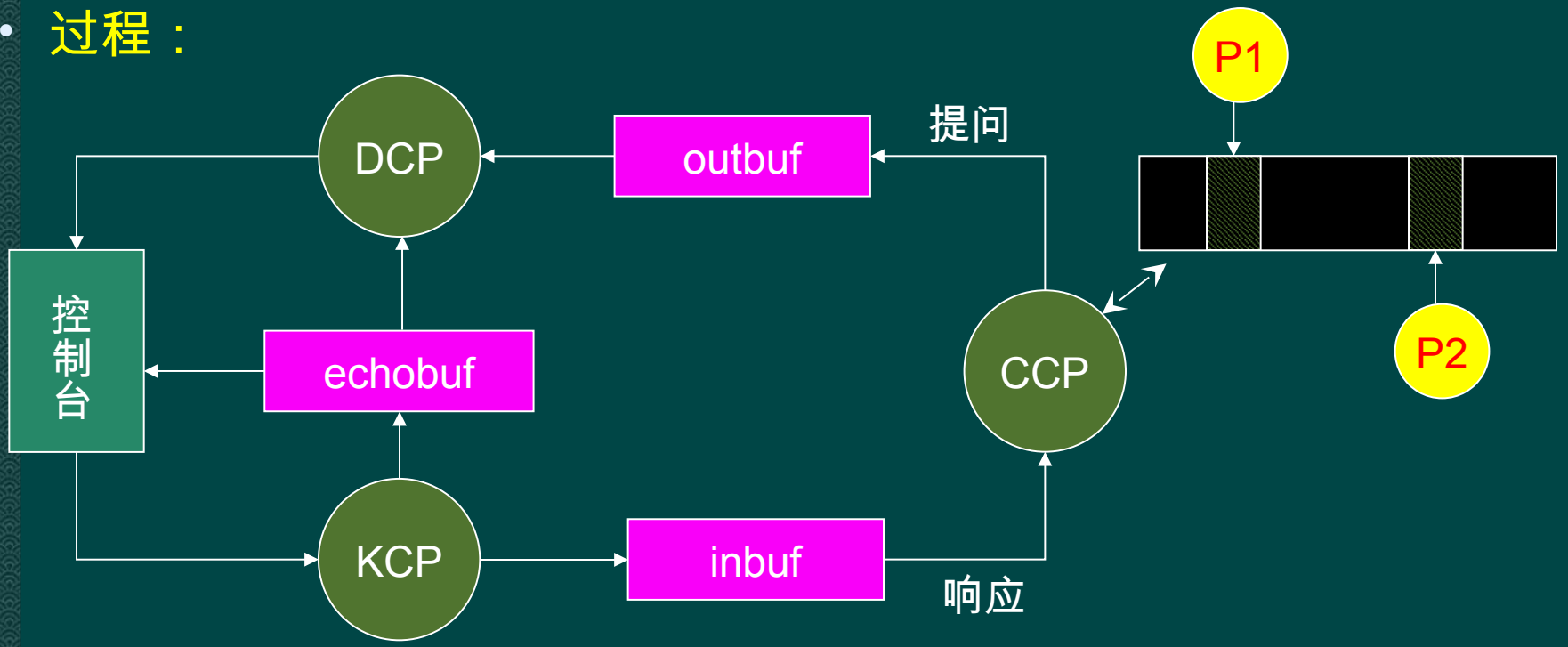


3.7 进程通信

- 邮箱通信
  - 对于只有一发送进程和一接收进程使用的邮箱，进程间通信应满足如下条件 ：
    - 发送进程发送消息时，邮箱中至少要有一个空格能存放该消息。
    - 接收进程接收消息时，邮箱中至少要有一个消息存在。

3.7 进程通信实例——控制台通信

- 控制台终端由：
  - **键盘控制进程 - - KCP**
  - **显示控制进程 - - DCP**
- 用户进程和控制台终端的通信由：
  - **会话控制进程 --CCP**
- 过程：



3.7 进程通信实例——控制台通信

- KCP 和 DCP 动作

设 K-data-Ready 和 K-data-Empty 分别为键盘 KP 和键盘控制进程 KCP 的私用信号量，其初值为 0 和 1 。

KCP 进程如下 :

初始化 { 清除所有 inbuf 和 echobuf}

```
begin local x
        P(K-data-Ready)
        从键盘数据传输缓冲 x 中取出字符 m 记为 x.m
        inbuf<=(x.m) // 将 x.m 送输入缓冲区
        echobuf <=(x.m)// 将 x.m 送入显示缓冲区
        V(K-data-Empty)
End
```

键盘动作 KP :

```
repeat local x
        P(K-data-Empty)
        把键入字符放入数据传输缓冲 x
        V(K-data-Ready)
Until 终端关闭
```

3.7 进程通信实例——控制台通信

◈ KCP 和 DCP 动作

显示器控制进程 DCP ：设 D-data-Ready 和 D-data-Empty 分别为 DP 和 DCP 的私用信号量且初值为 0 和 1 。

```
初始化 { 清除 outbuf,echo 模式置 false }
begin
        if outbuf 满
        then
            receive(k)/*CCP k*/
            P(D-data-Empty)
              把 k 送入显示器数据缓冲区
            V(D-data-Ready)
        else
                echo 模式置 true
                echobuf 中字符置入显示器数据缓冲区
        fi
end
```

显示器动作 DP ：

```
repeat
            if echo    模式
            then
                    打印显示器数据缓冲区中字符
            else
                  P(D-data-Ready)
                    打印显示器数据缓冲区中消息
                  V(D-data-Empty)
Until  显示器关机
```

3.7 进程通信实例——控制台通信

◈ CCP 和 KCP 及 DCP 的接口

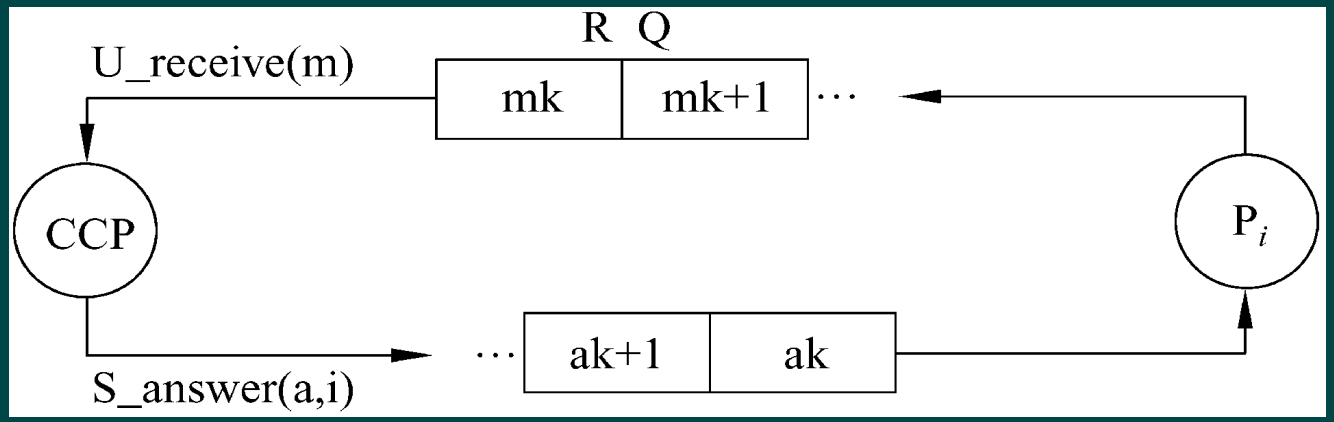
```
Read(x) ：  /* 把 inbuf 中的所有字符读到用户进程数据区 x 处 */
Begin
        P(inbuf-full)
        Copy(inbuf into x)
        V(inbuf-empty)
end
```

```
Write(y) ：  /* 把用户进程 y 处的消息写到 outbuf 中 */
Begin
        P(outbuf-empty)
        Copy(outbuf from y)
        V(outbuf-full)
end
```

- inbuf-full 和 inbuf-empty 分别是 CCP 和 KCP 的私用信号量，在过程 Send(m) 和 Read(x) 中使用，其初值分别为 0 和 1 。
- outbuf-full 和 outbuf-empty 则是 DCP 和 CCP 的私用信号量，在过程 receive(k) 和 write(y) 中使用，其初值分别为 0 和 1 。

3.7 进程通信实例——控制台通信

- CCP 和用户进程的接口
  - 设备用户进程向 CCP 发出的提问用消息组成队列 RQ 。各用户进程把消息送入 RQ 时，必须互斥操作，否则将引起 RQ 队列混乱。因此，设互斥用信号量 rq ，初值为 1 。
  - CCP 只有在用户进程提问之后才负责向控制台转发提问和向用户进程转达控制台的指示。因此，我们还必须为 CCP 设置一私用信号量 question 以计算用户进程所提出的问题数。信号量 question 的初值为 0 。
  - 由于各用户进程在 CCP 发出回答消息之后，不一定马上就能处理 CCP 所发出的回答消息，因而，需设置相应的消息接收队列 SQi 。 SQi 和 RQ 的关系如图：



```
receive(m):
    begin
      P(question)
      when rq do 从 RQ 中取出
      Return(m)
    end //rq 为临界区名，RQ 为提问队列

answer(a,i)
   begin
      when sqi do 把 a 插入到 SQI 中
      V(answer i)
   end  //sqi 为临界区名，SQI 为答案队列

   CPP 会话进程如下：
     local k,m,x,y
     repeat receive(m)
                  将消息 m 的进程号置入 k 中；将消息 m 解码变换到 x
            Write(x);
            Read(y);
              将 y 编码到 m
            answer(m,k)
     until CCP 结束
```

3.7 进程通信实例——管道

管道 pipe

- 进程通信的使用例子之一是 UNIX 系统的管道通信。 UNIX 系统从 System V 开始，提供有名管道和无名管道两种通信方式，这里介绍无名管道。
- 无名管道为建立管道的进程及其子孙提供一条以比特流方式传送消息的通信管道。
- 管道在逻辑上被看作管道文件，在物理上则由文件系统的高速缓冲区构成 。
- 管道按 FIFO （先进先出）方式传送消息，且只能单向传送消息。

3.7 进程通信实例——管道

- 关于管道的操作：
  - UNIX 提供的系统调用 pipe ，可建立一条同步通信管道。其格式为：
    - int fd[2];
    - pipe(fd)

    这里， fd[1] 为写入端， fd[0] 为读出端。

3.7 进程通信实例——管道

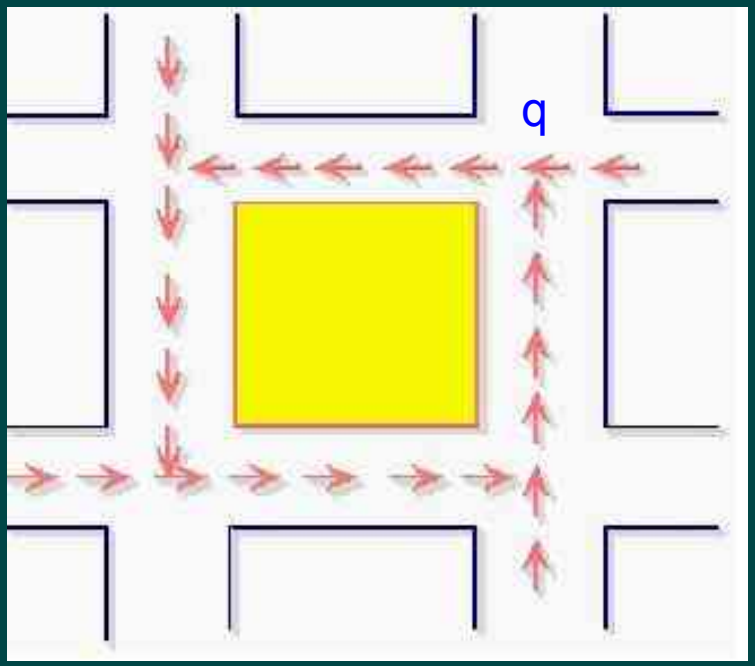
例 1 ：用 C 语言编写一个程序，建立一个 pipe ，同时父进程生成一个子进程，子进程向 pipe 中写入一字符串，父进程从 pipe 中读出该字符串。

```
# include <stdio.h>
main()
{
        int x, fd[2];
        char buf[30],s[30];
        pipe(fd);  /* 创建管道 */
        while((x=fork()) = = -1); /* 创建子进程失败时，循环 */
         if(x = = 0){
                sprintf(buf,  ”This is an example\n”);
                write(fd[1],buf,30); /* 把 buf 中字符写入管道 */
                exit(0);
         }
          else{
                wait(0);
                read(fd[0],s,30);  /* 父进程读管道中字符 */
                printf(“%s”,s);
         }
  }
```


2005.11.04

3.8 死锁问题

- 死锁定义
  - 所谓死锁，是指各**并发进程彼此互相等待对方所拥有的资源**，且这些并发进程在得到对方的资源之前不会释放自己所拥有的资源。



3.8 死锁问题

- 死锁起因
  - 计算机系统中存在**许多独占资源**，如
    - 硬件资源：磁带机、绘图仪等
    - 软件资源：进程表、临界区等
  - 为保证有效使用独占资源，必须经历：
    - **申请资源**，若资源不可用，则申请者进入等待状态
    - **使用资源**
    - **归还资源**
  - 当一个进程需要独占多个资源，而**操作系统允许多个进程并发共享系统资源**时，可能会出现进程永远处于等待状态的现象（死锁）

3.8 死锁问题

- 产生死锁的必要条件：
  - **互斥**条件：并发进程所要求和占有的资源是不能同时被两个以上进程使用或操作的，进程对它所需要的资源进行排它性控制。
  - **不可剥夺**条件：进程所获得的资源在未使用完毕之前，不能被其它进程强行剥夺，而只能由获得该资源的进程自己释放。
  - **部分分配（占有还需）**：进程每次申请它所需要的一部分资源，在等待新资源的同时，继续占用已分配到的资源。
  - **环路**条件：存在一种进程循环链，链中每一个进程已获得的资源同时被下一个进程所请求。

3.8 死锁问题

- 死锁的排除方法

方法	资源分配策略	各种可能模式	主要优点	主要缺点
预防 Prevention	保守的；宁可资源闲置	1. 一次请求所需全部资源； 2. 资源剥夺； 3. 资源按序申请.	• 适用于作突发式处理的进程，不必剥夺； •适用于状态可以保存和恢复的资源； •可以在编译时（而不必在运行时）就进行检查	•效率低，进程初始化时间延长； • 剥夺次数过多，多次对资源重新起动； •不便灵活申请新资源
避免 Avoidance	是“预防”和“检测”的折衷	寻找可能的安全的运行顺序	不必进行剥夺	必须知道将来的资源需求；进程可能会长时间阻塞
检测 Detection	宽松的；只要允许，就分配资源	定期检查死锁是否已经发生	不延长进程初始化时间；允许对死锁进行现场处理	通过剥夺解除死锁，造成损失

3.9 线程

- 线程的引入：

  引入**进程**的目的是为了使多个程序并发执行，以改善资源利用率、提高系统吞吐量。

  引入**线程**则是为了减少进程并发执行时所付出的上下文切换开销。

- 线程的定义：
  - 有时称轻量级进程。 进程中的一个运行实体，是一个 CPU 调度单位。
  - 线程自己基本不拥有系统资源，只拥有少量必不可少的资源：程序计数器、一组寄存器、栈。
  - 它可与同属一个进程的其它线程共享进程所拥有的全部资源。
  - 一个线程可以创建和撤消另一个线程；同一进程中的多个线程之间可以并发执行。

3.9 线程

- 线程与进程区别
  - **调度**
    - 在传统 OS 中，拥有资源、独立调度和分派的基本单位都是进程，在引入线程的系统中，**线程是调度**和分派的基本单位，而**进程是拥有资源**的基本单位。
    - 在同一个**进程内线程切换不会产生进程切换**，由一个进程内的线程切换到另一个进程内的线程时，将会引起进程切换。
  - **并发性**
    - 在引入线程的系统中，进程之间可并发，同**一进程内的各线程之间也能并发执行**。因而系统具有更好的并发性。

3.9 线程

- **线程与进程区别**
  - 拥有资源：无论是传统 OS ，还是引入线程的 OS ，进程都是拥有资源的独立单位，线程一般不拥有系统资源，但它可以访问隶属进程的资源。即一个进程的所有资源可供进程内的所有线程共享。
  - 系统开销：进程的创建和撤消的开销要远大于线程创建和撤消的开销，进程切换时，当前进程的 CPU 环境要保存，新进程的 CPU 环境要设置，线程切换时只须保存和设置少量寄存器，并不涉及存储管理方面的操作，可见，进程切换的开销远大于线程切换的开销。
  - 同时，同一进程内的各线程由于它们拥有相同的地址空间，它们之间的同步和通信的实现也变得比较容易。

3.9 线程

- 线程适用范围
  - 服务器中的文件管理或通信控制
  - 前后台处理
  - 异步处理
  - 数据的批处理
  - 网络系统中信息的并发处理

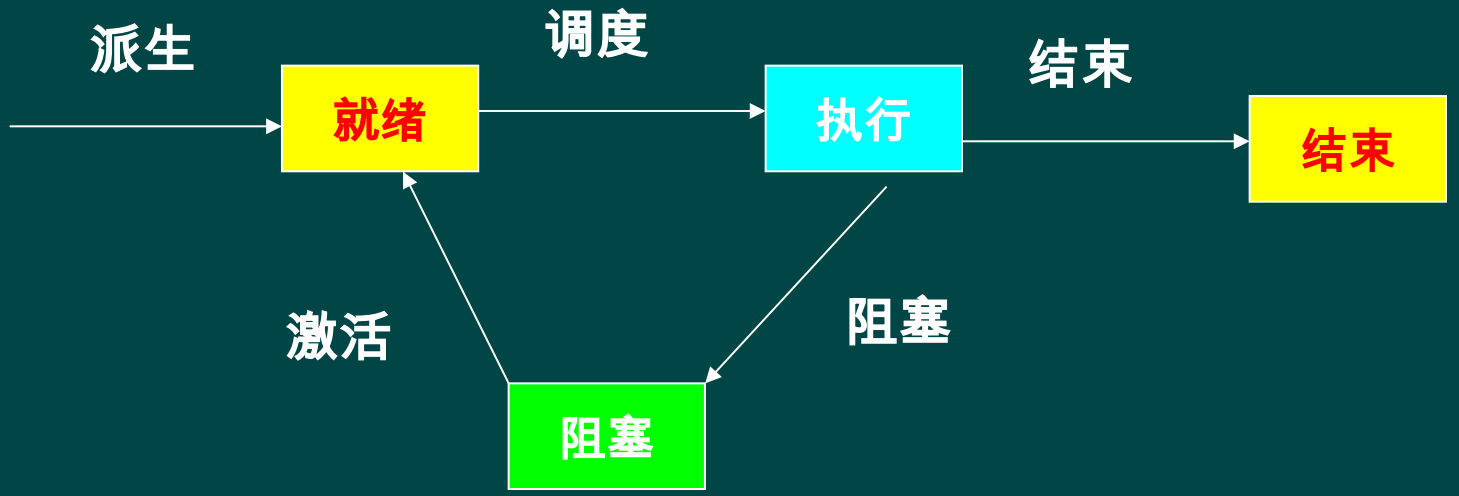
3.9 线程

◈ 线程的分类

分类	定义	优点	缺点
用户级线程（ULT）	管理过程全部由用户程序完成，操作系统内核心只对进程进行管理	•线程切换不调用内核 •调度是应用程序特定的 •ULT 可运行在任何操作系统上（只需要线程库），可以在一个不支持线程的 OS 上实现	•大多数系统调用是阻塞的，因此核心阻塞进程，故进程中所有线程将被阻塞 •核心只将处理器分配给进程，同一进程中的两个线程不能同时运行于两个处理器上
系统级线程	由操作系统内核进行管理。操作系统内核提供相应的系统调用和 API，使用户程序可以创建、执行、撤消线程。	•对多处理器，核心可以同时调度同一进程的多个线程 •阻塞是在线程一级完成	•在同一进程内的线程切换调用内核，导致速度下降

3.9 线程

- 线程的执行特性
  - 线程有 3 个基本状态：执行、就绪、阻塞
  - 线程有 5 种基本操作：派生、阻塞（ Block ）、激活（ unblock ）、调度（ schedule ）、结束（ Finish ）



3.9 线程

- 线程的另一个执行特性是同步。**线程中所使用的同步控制机制**与进程中所使用的同步控制机制相同。

小 结

- 进程是系统分配资源的基本单位。
- 进程的静态描述
- 进程具有动态性、并发性等特点。存在同步互斥问题
- 进程间的通信
- 线程是最小运行实体

结束