# 第五章 同步

## 5.2 临界区问题

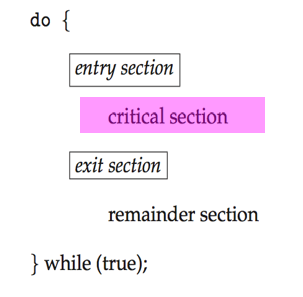
**1、临界资源**

一次仅允许一个进程访问的资源

**2、临界区**

进程中访问临界资源的一段代码

**3.临界区的程序结构**



**4、临界区进入原则（简答题、选择题）**

①互斥：

如果一个线程在临界区中执行，则其他线程不允许进入

②有空让进：

若干线程要求进入空闲临界区时，应尽快安排一个线程进入

③有限等待：

从线程发出请求到允许进入，不能无限等待

## 5.3 Peterson方案

**1、软件方法**：两个线程交替访问临界区的软件方法

2、结合了标记法和轮转法

3、满足临界区原则：互斥、有空让进、有限等待

## 5.4 硬件互斥

**一、禁止中断**

**1、适用于：**

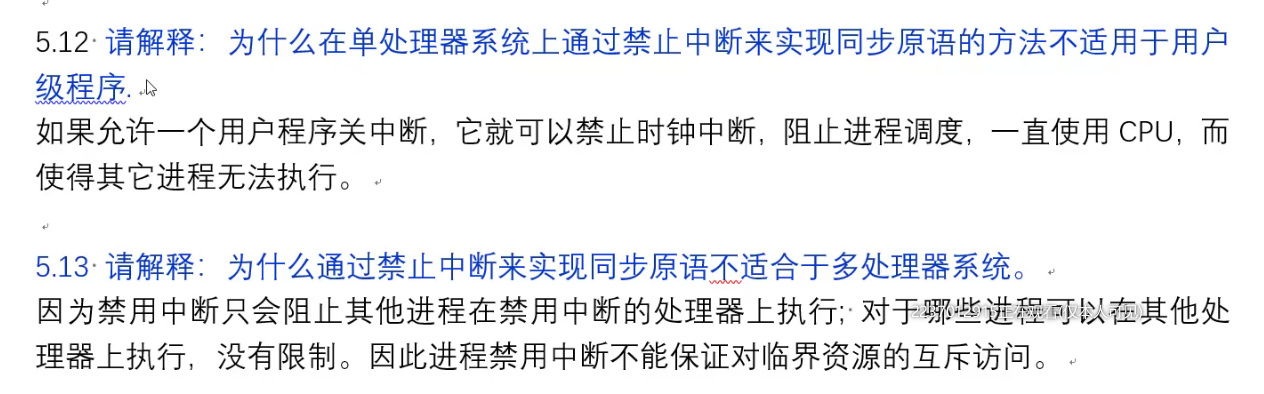
**单处理器系统**，**不适合于用户进程**

2、关中断后，时钟中断也被屏蔽，而进程调度需要使用到系统时钟，所以在同一CPU中不会调度其他线程

**3、缺点：**

关中断时间过长，系统效率低，限制CPU交叉执行程序的能力

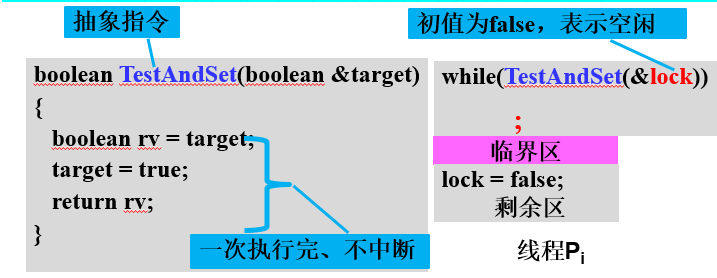
**4、简答题**



**二、硬件原子指令**

1、特殊硬件指令、不可中断

2、



**3、缺点**：

①复杂易出错

②循环测试可能导致**饥饿和死锁**

**4、适合于**：

单处理器系统Or多处理器系统

5、OS提供的临界区互斥方法（由系统而不是用户实现）：

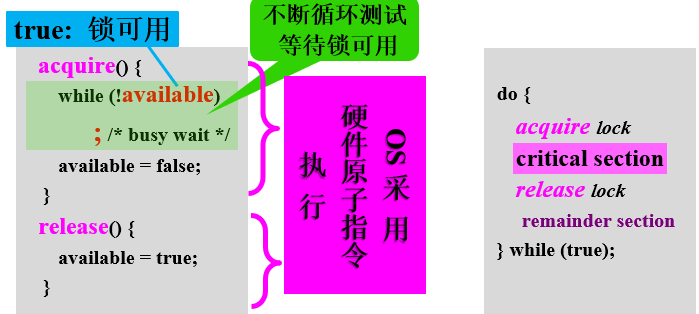
①软件：Peterson

②关中断

③硬件原子指令

## 5.5 自旋锁和互斥锁

**一、自旋锁**



1、优点：

没有上下文切换的开销

2、缺点：

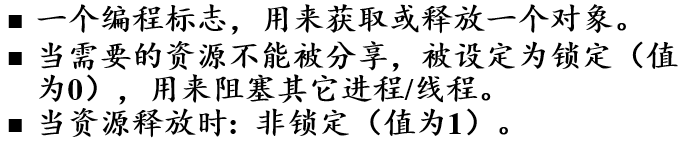
“忙等待“，加锁全程消耗CPU

3、适用：

临界区持锁时间短、多CPU系统

**二、互斥锁**

1、



2、互斥量

信号量的简化版本（没有技术能力），只有2种状态：加锁和解锁

**三、自旋锁和互斥锁的区别**

①等待方式不同：

自旋锁：忙等待，不断循环测试锁的状态，一直占用CPU；

互斥锁：放弃CPU—进入阻塞状态—唤醒阻塞的进程

②适用于不同地方：

自旋锁：临界区持锁时间短、多CPU系统

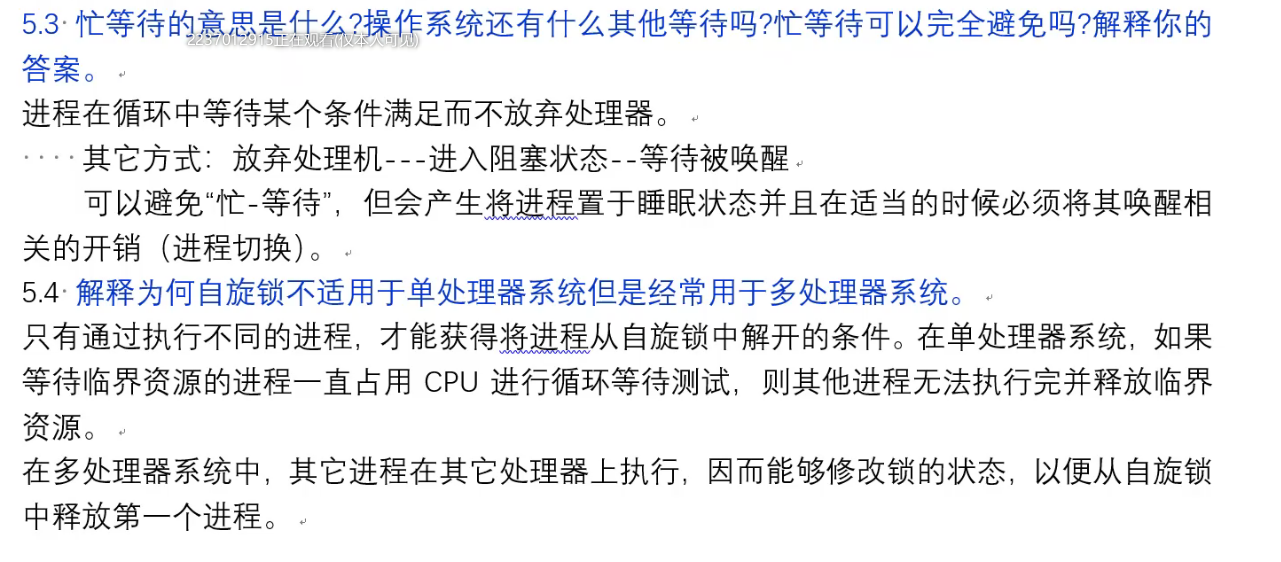
互斥锁：临界区持锁时间长

③开销的比较：

自旋锁：起始开销小，但临界区持锁时间越长，开销增加剧烈

互斥锁：起始开销大（涉及到用户态/内核态的转换），但与临界区持锁时间无关

**四、简答题**

****

## 5.6 信号量

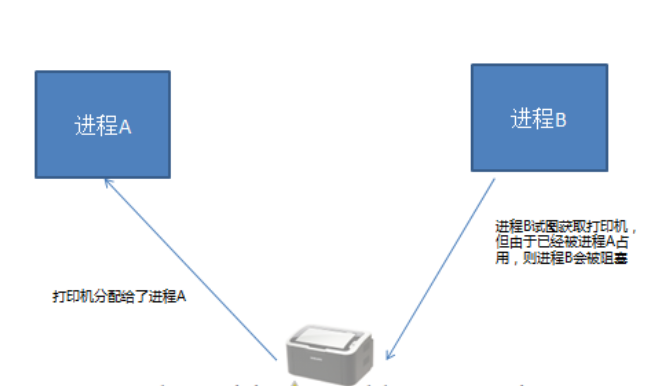
**一、并发进程的制约关系**

**1、间接制约（互斥）**

①竞争**共有资源**，临界区互斥

②竞争没有联系（A、B线程都要向屏幕输出）

③举例：

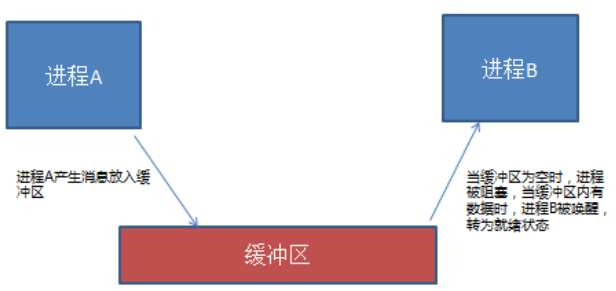


**2、直接制约（同步）**

①一组并发线程，因直接制约而相互合作，使得线程按一定的顺序和速度执行的过程

②共享对方的私有资源

③之间有联系、合作关系



**二、信号量的物理意义**

**1、信号量**

①一个整数值

②物理含义：

S>0: 有S个资源可用

S==0: 没有资源可用

S<0: 绝对值表示等待队列中的进程个数（与可用资源个数无关）

③可执行的操作

初始化（非负）、PV操作（标准原子操作）

**2、原语**

①定义：

系统态下执行的具有特定功能的程序段

②分类：

机器指令级：不允许中断

功能级：不允许并发执行

③举例：

进程的创建、撤销、唤醒等

PV操作（原子指令）

**三、公用信号量和私用信号量**

1、公用信号量（**实现互斥**）

①初值为1，每个线程均可施加P、V操作

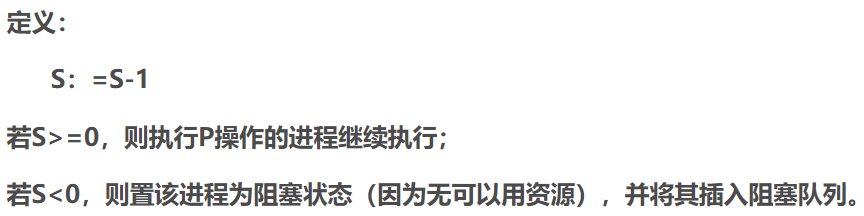
2、私用信号量（**实现同步**）

①初值为0或正整数，表示资源的数目

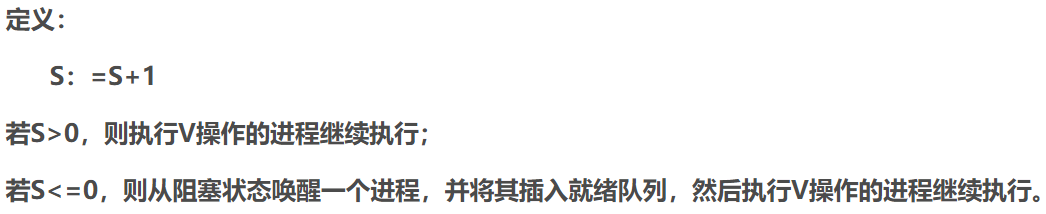
**②P操作的是自己的私有信号量、V是操作的别人的私有信号量**

**四、信号量的实现**

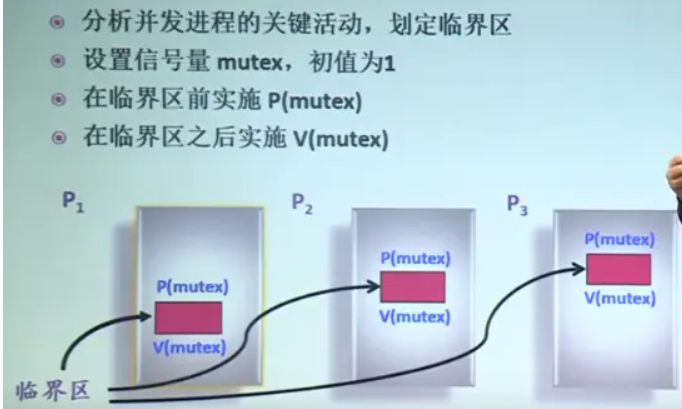
**1、P操作（申请一个资源）**

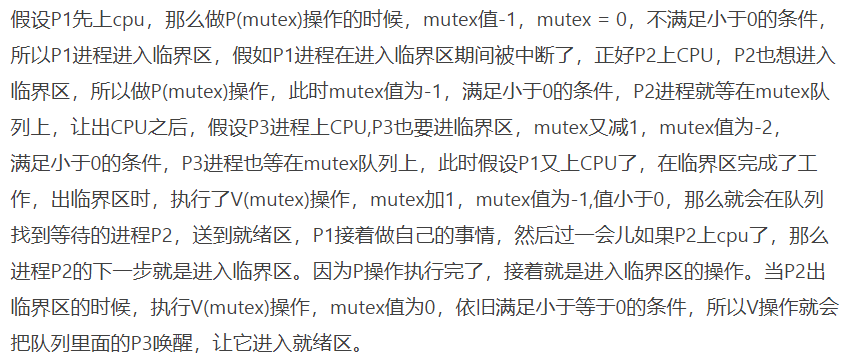


**2、V操作（释放一个资源）**



**3、PV操作举例**





**4、PV操作易产生的疑问**

①**S大于0那就表示有临界资源可供使用，为什么不唤醒进程？**

答：S大于0的确表示有临界资源可供使用，也就是说这个时候没有进程被阻塞在这个资源上，所以不需要唤醒。

②**S小于0应该是说没有临界资源可供使用，为什么还要唤醒进程？**

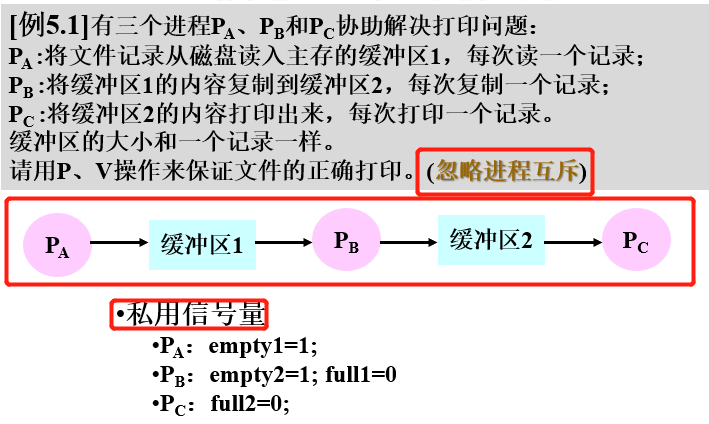
答：V原语操作的本质在于：一个进程使用完临界资源后，释放临界资源，使S加1，以通知其它的进程，这个时候如果S<0，表明有进程阻塞在该类资源上，因此要从阻塞队列里唤醒一个进程来“转手”该类资源。比如，有两个某类资源，四个进程A、B、C、D要用该类资源，最开始S=2，当A进入，S=1，当B进入S=0，表明该类资源刚好用完， 当C进入时S=-1，表明有一个进程被阻塞了，D进入，S=-2。当A用完该类资源时，进行V操作，S=-1，释放该类资源，因为S<0，表明有进程阻塞在该类资源上，于是唤醒一个。

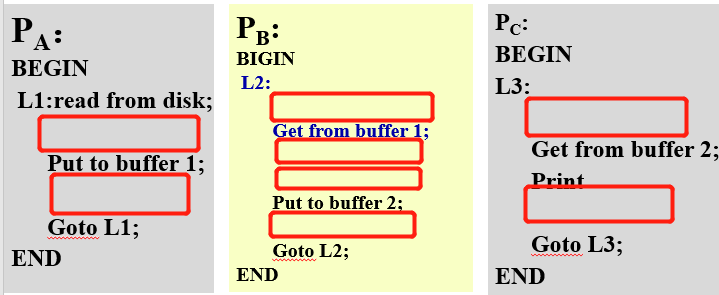
③**如果是互斥信号量的话，应该设置信号量S=1，但是当有5个进程都访问的话，最后在该信号量的链表里会有4个在等待，也是说S=-4，那么第一个进程执行了V操作使S加1，释放了资源，下一个应该能够执行，但唤醒的这个进程在执行P操作时因S<0，也还是执行不了，这是怎么回事呢？**

**答：**当一个进程阻塞了的时候，它已经执行过了P操作，并卡在临界区那个地方。当唤醒它时就立即进入它自己的临界区，并不需要执行P操作了，当执行完了临界区的程序后，就执行V操作。

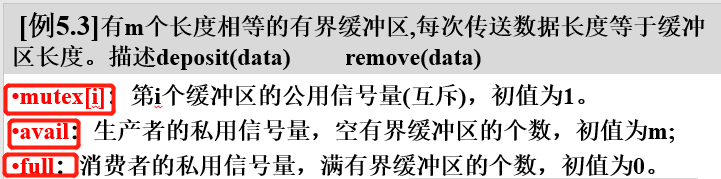
## 5.7 用信号量解决同步问题

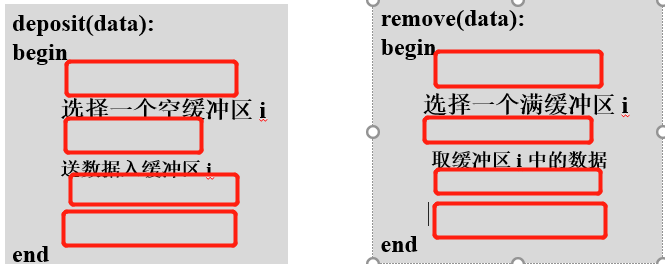
**1、**





**2、**





**3、P、V操作讨论**

①P、V操作必须成对出现

②互斥操作：处于同一进程

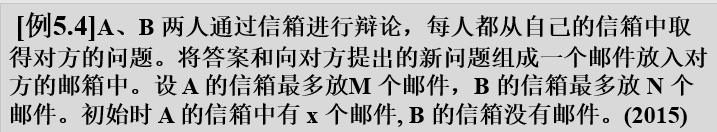
③同步操作：不在同一进程中

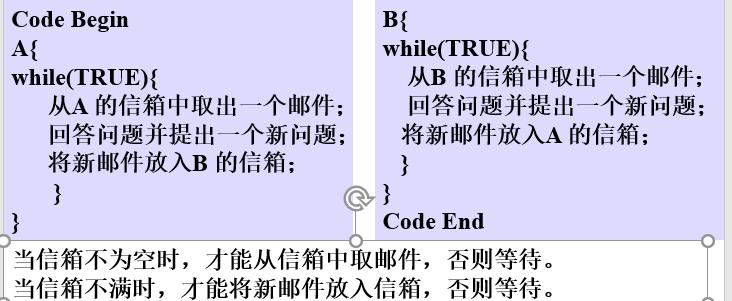
④2个P操作在一起，顺序至关重要：

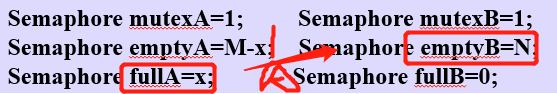
同步P操作和互斥P操作在一起：同步P在前；否则将导致死锁

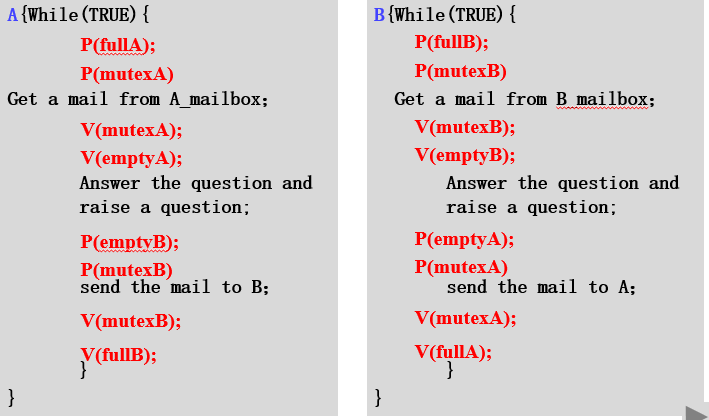
两个V操作顺序无关紧要

**4、**





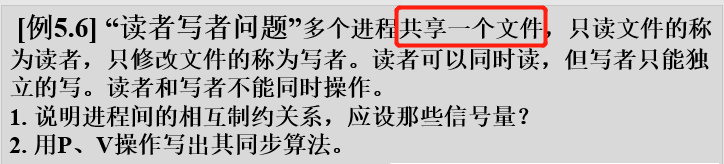




5、哲学家问题

6、读者写者问题

**①问题描述：**



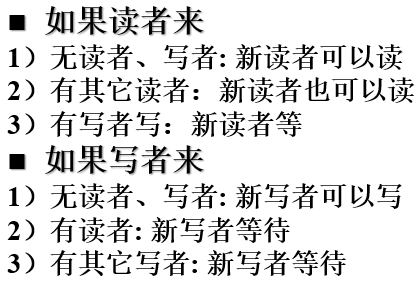
总结下就是：

写着向数据区放数据，读者从数据区获取数据

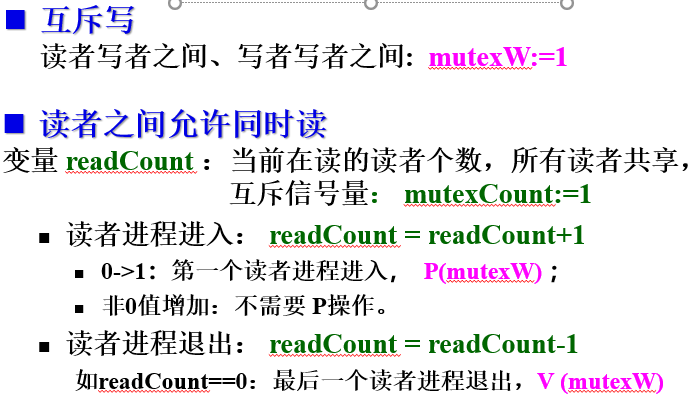
多个读者可以同时读取数据

多个写者不能同时写数据

读者和写者不能同时操作



**② 解法1.0：**

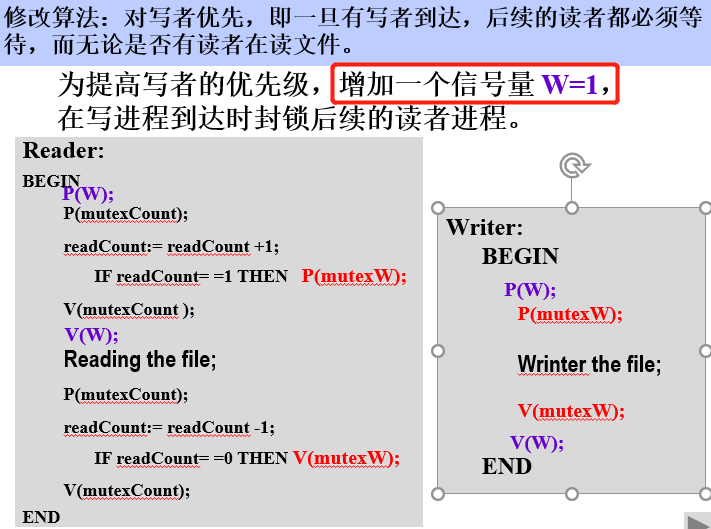




这个算法的缺陷：

如果一直有读者在读，写者没有机会。导致系统中写者的权力被弱化，文件无法更新。

**③ 解法2.0：**



其中，V(W)的位置必须放在Reading the file 前面，否则将导致不能实现同时有多个读者可以读文件这一要求。

**7、信号量的优缺点：**

①优点：

简单；

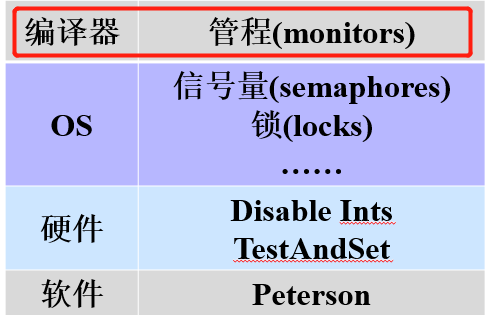
表达能力强：利用PV操作可以解决任何同步互斥问题

②缺点：

不安全，使用不当容易出现死锁问题

遇到复杂同步互斥问题实现复杂

## 5.8 管程



## 5.10 死锁

**1、死锁概述**

**① 死锁的定义**

一组并发进程彼此互相等待对方所拥有的资源，在得到对方的资源之前不释放自己所拥有的资源，若无外力驱动，各进程不能向前推进的僵死状态。

**② 饥饿的定义**

某个进程可能永远在信号量的阻塞队列中

**③ 死锁的原因：**

Ⅰ根本原因：并发进程的资源竞争（不可抢占的资源）

Ⅱ进程推进顺序不当

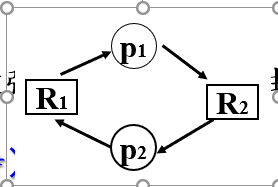
**④ 死锁的必要条件**

互斥：涉及的资源是临界性的

不可抢占：该进程占用的资源，不会被强制性拿走，必须由该进程主动释放

部分分配：进程在等待新资源时继续占有已分配的资源

环路等待：存在一种进程的循环链，链中的每一个进程已获得资源同时被下一个进程所请求。



**⑤ 死锁的处理方法**

Ⅰ不让死锁发生

死锁预防：破坏死锁的必要条件

动态避免：

Ⅱ让死锁发生

检测并恢复

Ⅲ鸵鸟政策🡪交给程序设计人员来解决，绝大多数操作系统都采用这种方法

**⑥ 死锁预防**

实现：打破互斥条件，将不可共享的变成可共享的

例子：分时CPU、虚拟合并

**⑦ 死锁避免**

Ⅰ安全状态的定义：

自身不存在死锁；

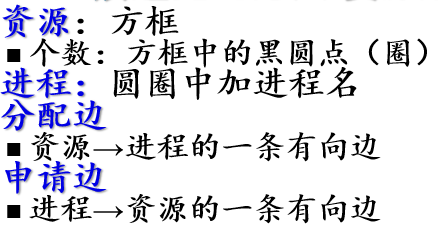
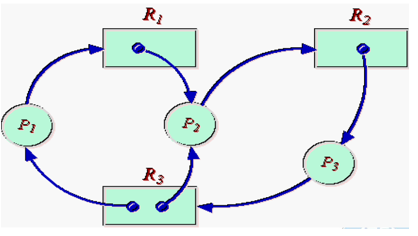
存在着某种调度顺序，即使在最坏的情况下即所有的进程请求他们最大数目的资源，每一个进程都能够顺利地运行结束。

Ⅱ不安全状态有可能会导致死锁；安全状态一定不会导致死锁

Ⅲ死锁避免的方法

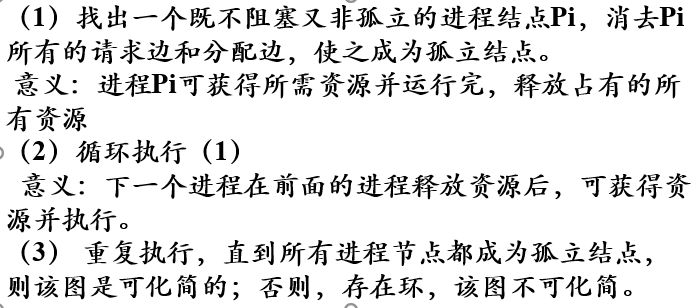
银行家算法：理论上精彩的，实际上无用的

资源分配图



死锁定理（死锁状态的充分条件）：资源分配图不可完全简化

资源图的化简：



**⑧ 死锁的检测和解除**

检测：

定期启动一个软件检测系统的状态，若发现有死锁存在，则采取措施

死锁检测算法：确定是否存在环路，资源分配图和死锁定理

解除：

剥夺资源、进程回滚、撤销资源