操作系统

Operating System

第二章

进 程 管 理

第二章 进程管理

进程是OS最基本最重要的概念，进程管理是OS的重点和难点。

什么是进程？为什么要引入进程？

首先看一个与进程有关但不相同的概念：

程序：指令的有序集合。

指令的种类和顺序描述了程序的功能。

2·1 进程概念的引入

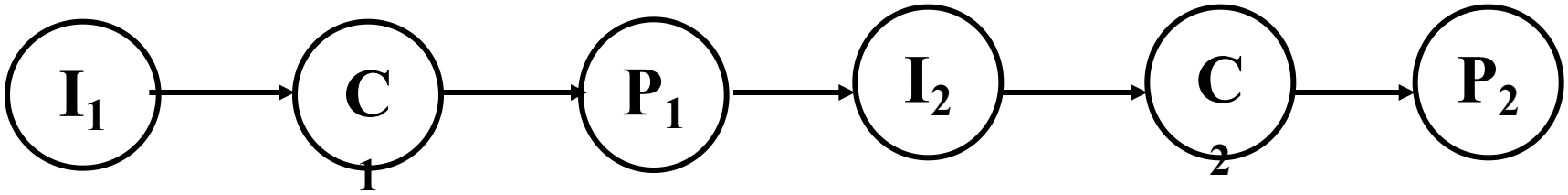
1、程序的执行

程序的执行有两种方式：顺序执行和并发执行。

- 顺序执行是单道批处理系统的执行方式，也用于简单的单片机系统；
- 现在的操作系统多为并发执行，具有许多新的特征。引入并发执行的目的是为了提高资源利用率。

2·1 进程概念的引入

1）程序的顺序执行：



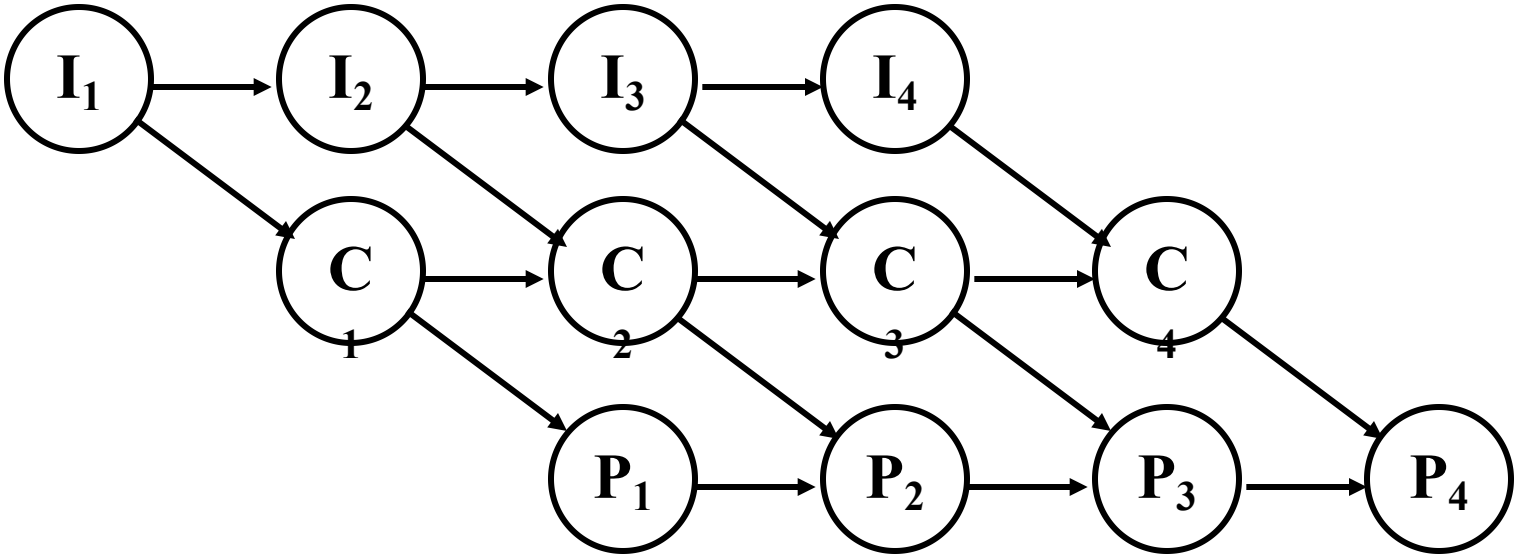
【特征】：

- ☞顺序性：按照程序结构所指定的次序（可能有分支或循环）
- ☞封闭性：独占全部资源，计算机的状态只由于该程序的控制逻辑所决定
- ☞可再现性：初始条件相同则结果相同。

在单道环境中，程序的执行具有封闭性和可再现性的特点，程序的概念足以描述程序的执行过程。（pc指令计数器的值可以描述程序执行到哪里，计算机的状态的改变封闭于正在执行的程序）

2·1 进程概念的引入

2）程序的并发执行：



【并发执行】：在多道程序下，任意时刻系统中有多个活动并发执行。这是现代OS的一个基本特征。

【资源共享】：统中的硬件资源和软件资源由多道用户程序共同使用，资源的状态由多道程序所决定。这是现代OS的另一个基本特征。

2·1 进程概念的引入

【并发执行特征】：

- 间断(异步)性：“走走停停”，一个程序可能走到中途停下来，失去原有的时序关系；
- 失去封闭性：共享资源，受其他程序的控制逻辑的影响。如：一个程序写到存储器中的数据可能被另一个程序修改，失去原有的不变特征。
- 失去可再现性：程序与CPU执行的活动之间不再一一对应，程序经过多次运行，虽然其各次的环境和初始条件相同，但得到的结果却各不相同。
- 相互作用和制约性：系统中并发执行的程序具有相互独立的一面（表现在每个程序为用户提供特定的功能，它们之间相互独立），但是有时也会直接或间接的发生相互依赖和相互制约。

在多道环境下，程序的执行产生了新的特点（失去封闭性，不可再现），原来程序的概念不足以描述多道程序的并发执行。

例如，在任一时刻：

- 哪道程序在cpu上运行?
- 该程序在运行过程中已经使用了那些资源？为了运行完毕还需要哪些资源?
- 不在cpu上运行的程序是什么原因等待?
- 每一道程序装在内存中的什么地方?

。。。。。。

操作系统作为覆盖在裸机上的第一层软件，必须合理的组织计算机工作流程，协调多道程序的运行，对以上问题必须能够了如指掌。显然，程序的概念无法胜任多道环境下的新特点，由此引入进程的概念。

2·1 进程概念的引入

1、进程的概念：

进程是操作系统中一个最基本也是最重要的概念，但目前没有一个非常确切的定义。

为了强调进程并发性和动态性的特点，将其定义为：

进程是①程序关于某个②数据集合在③处理机上的④一次*执行过程*。

进程的两个属性：

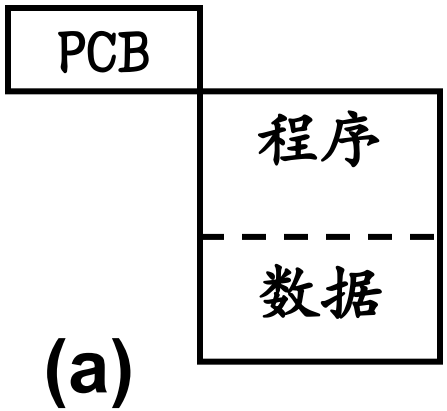
1、进程是操作系统进行资源分配的单位

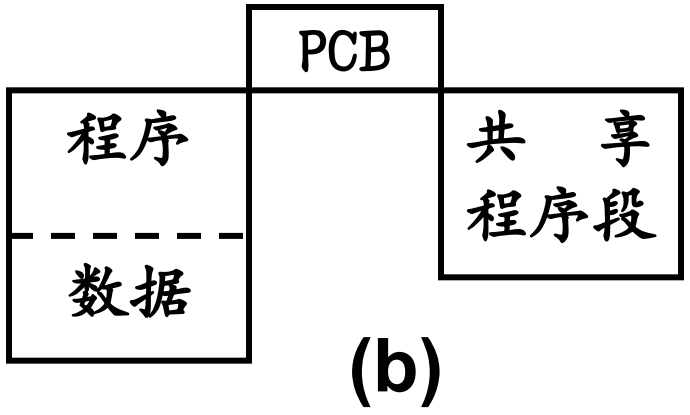
2、进程是操作系统进行处理机调度的单位

2·2 进程的表示

2、进程的组成：进程通常由三部分组成：

- ☞程序：描述了进程所要完成的功能，是进程执行时不可修改的部分。
- ☞数据集合：程序执行时所需要的数据和工作区，为一个进程专用，可修改。
- ☞进程控制块PCB(Process Control Block)：包含了进程的描述信息和控制信息，是进程的动态特性的集中反映。





2·2 进程的表示

3、进程控制块PCB(Process Control Block)：

为了对进程进行有效的控制和管理，系统为每一进程设置一个进程控制块（结构体变量），PCB其是进程存在的唯一标志。通常PCB包含以下几类信息：

1）进程描述信息：

2）进程控制信息：

3）资源占用信息：

4）CPU现场保护结构：

2.2 进程的表示

1）进程描述信息：

- 进程标识名(Process ID)：也称为标识符或标识数，为进程的内部标识，用来唯一标识一个进程，通常是一个整数。
- 进程名：为进程的外部标识，通常基于可执行文件名（不唯一）；

2）进程控制信息：

- 当前状态：进程当前所处状态，为进程调度之用。
- 优先级：进程需要处理的缓急程度标识。
- 程序和数据的地址：程序和数据所在的内存或外存地址。（十分重要！将进程的逻辑结构和组成实体联系起来）
- 队列指针或链接字：处于同状态的进程链接指针。

2.2 进程的表示

3）资源占用信息：

进程执行时除CPU外的资源的需求、分配和控制信息。

4）CPU现场保护结构：

它由处理机各种寄存器（通用寄存器、PC指令计数器、程序状态字PSW、用户栈指针等）的内容所组成，该类信息使进程被中断后重新执行时能恢复现场从断点处继续运行。

由于进程是描述程序运行的，不同的操作系统对进程的描述不尽相同，PCB结构体变量中包含的分量也各不相同，总而言之，把握一个原则：

所有与程序的运行有关的信息全部记录在PCB数据结构中。

3、进程控制块PCB的组织方式

PCB表:

系统把所有PCB组织在一起，并把它们放在内存固定区域，就构成了PCB表。

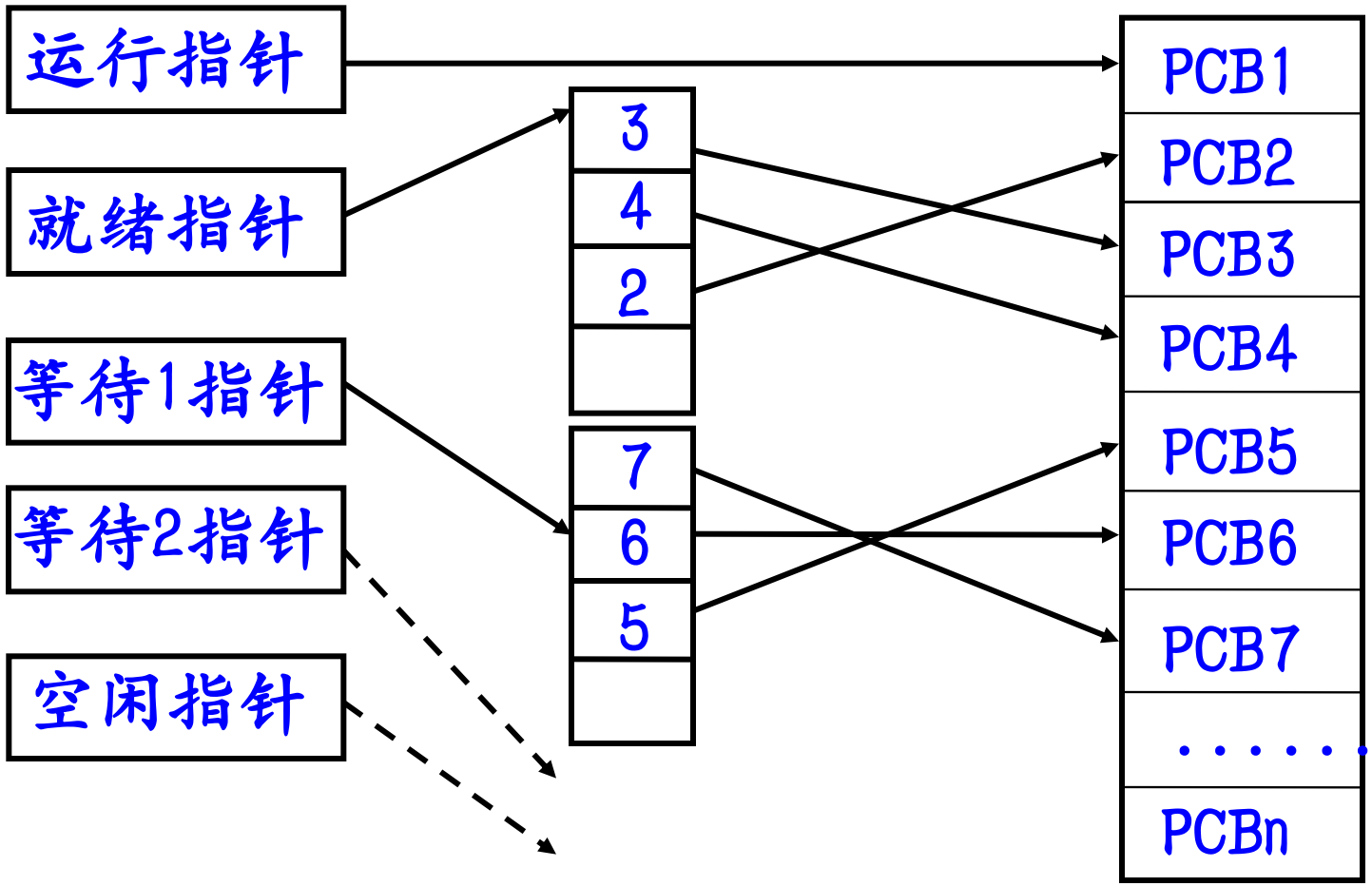
PCB表的大小决定了系统中最多可同时存在的进程个数，称为系统的并发度。

PCB表组织方式有两种：

索引方式和链接方式

1) 索引方式：

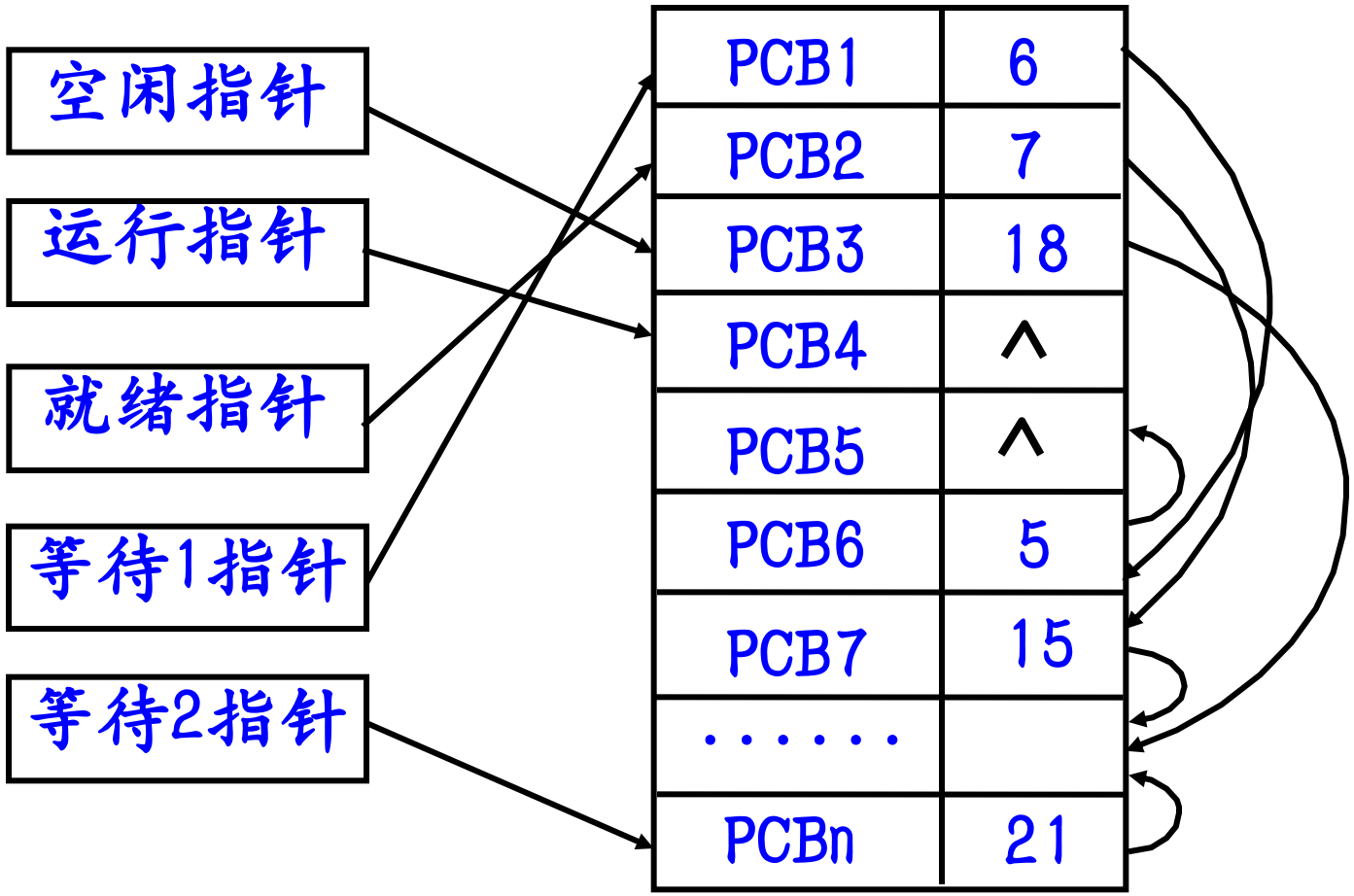
对具有相同状态的进程，分别设置各自的PCB索引表，表明PCB在PCB表中的地址



2) 链接方式：

对具有相同状态的进程，分别各自链接起来组成进程PCB链队列：

运行队列、就绪队列、阻塞队列、空闲队列



进程与程序的区别

比较方面	程序	进程
静态动态	程序是有序代码的集合，是静态的，通常对应着外存上的可执行文件	进程是程序的执行，是动态的
生存周期	永久性的	有生存周期的
组成	指令	程序、数据、PCB
对应关系	通过多次执行，一个程序可对应多个进程；	通过调用关系，一个进程可包括多个程序。

2·2 进程的表示

4、进程的特征

- **结构性：** 由程序+数据+进程控制块组成了进程实体，其中进程控制块是进程存在的标志。
- **动态性**进程是进程实体的执行过程，它由创建而产生，由调度而执行,因某事件而暂停，由撤销而消亡。
- **并发性：** 多个进程同时存于内存中,一起向前推进,并发执行。
- **独立性：** 进程是独立获得资源和独立调度的基本单位。
- **异步性：** 各进程都各自独立的不可预知的速度向前推进。

2.3 进程的状态

进程是一个动态的概念。如何表现出进程的动态性？为进程划分状态，通过状态的变化来表现其动态性。

进程状态的划分要求：

1、确定性：在进程生存周期中的任意时刻，进程的状态一定是唯一确定的。

2、充分性：进程的状态划分足以描述进程生存周期中的任意时刻。

2.3 进程的状态

1、进程的三种基本状态及其转换：

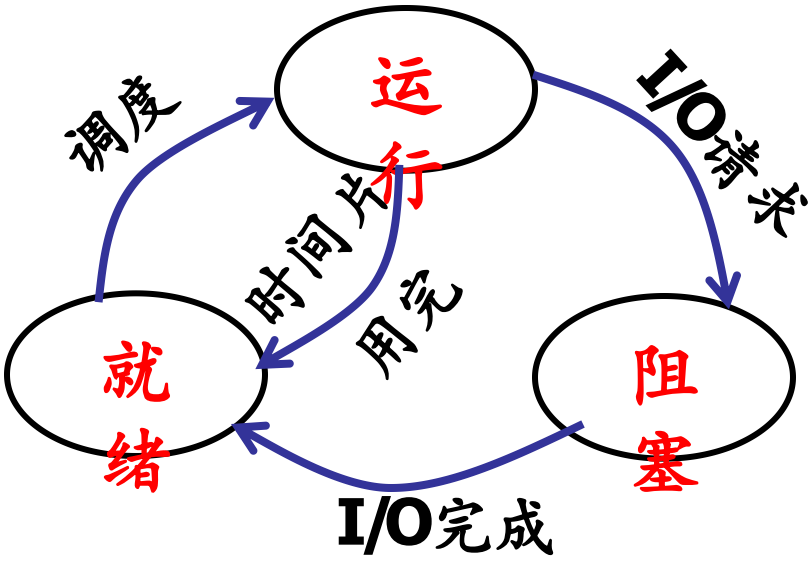
进程在生命期内处于且仅处于三种基本状态之一：

- ☞运行态（Running）：当一个进程在处理机上运行时，则称该进程处于运行状态。
- ☞就绪态（Ready）：一个进程获得了除处理机外的一切所需资源，一旦得到处理机即可运行，则称此进程处于就绪状态。
- ☞阻塞态（Blocked）：当一个进程正在等待某一事件发生（例如请求I/O而等待I/O完成等）而暂时停止运行，这时即使把处理机分配给进程也无法运行，故称该进程处于阻塞状态。注意与就绪状态的不同在于即使处理机处于空闲状态也无法运行。

2.3 进程状态

2、【状态转换】：

在进程运行过程中，由于进程自身进展情况及外界环境的变化，这三种基本状态可以依据一定的条件相互转换：



① 就绪→运行：调度程序选择一个新的进程运行

② 运行→就绪：运行进程用完时间片被中断或在抢占调度方式中，因为一高优先级进程进入就绪状态

③ 运行→阻塞：进程发生I/O请求或等待某事件时

④ 阻塞→就绪：当I/O完成或所等待的事件发生时

2.3 进程状态

3、细分的进程调度状态——挂起状态：

由于终端用户及操作系统的需要(排除故障或为系统减负)，为了能够将指定进程**暂时静止**下来，增加了**静止阻塞** (阻塞挂起) 和**静止就绪** (就绪挂起)态，原阻塞和就绪改称为**活动阻塞**和**活动就绪状态**。

①运行或活动就绪→静止就绪，活动阻塞→静止阻塞

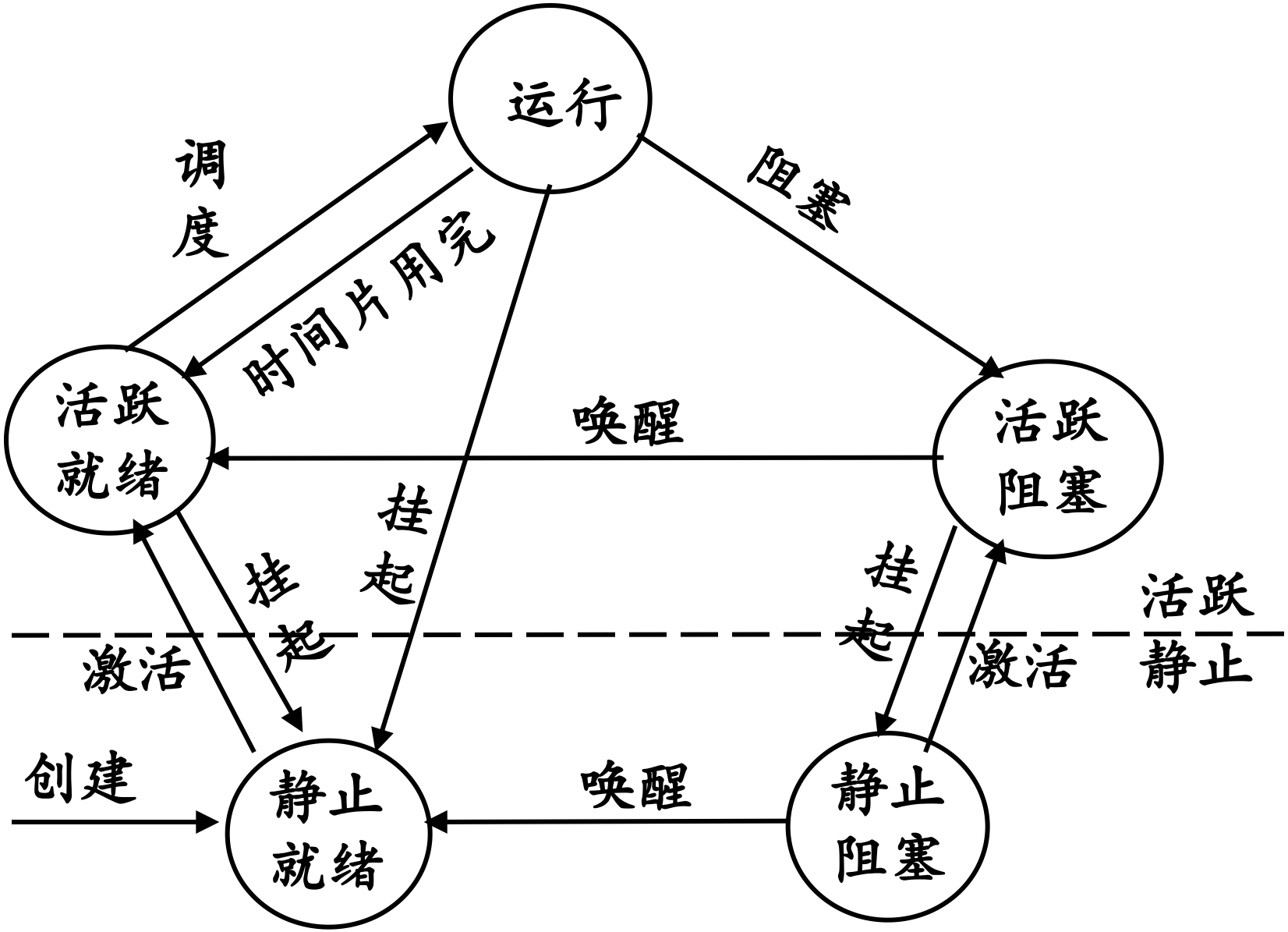
通过挂起操作(suspend)。

②静止就绪→活动就绪，静止阻塞→活动阻塞

通过激活操作(activate)。

③静止阻塞→静止就绪： 当等待的事件发生时。

具有挂起操作的进程状态转换图



课堂练习 2.1

在操作系统中进程是一个具有一定独立功能程序在某个数据集合上的一次__A__，进程是一个__B__概念，而程序是一个__C__的概念。在一单处理机中，若有5个用户进程，在非管态的某一时刻，处于就绪状态的用户进程最多有__D__个，最少有__E__个。

A：(1)并发活动　(2)运行活动
(3)单独操作　(4)关联操作

B，C：(1)组合态 (2)关联态 (3)运行态
(4)等待态 (5)静态 (6)动态

D，E：(1)1；(2)2；(3)3；(4)4；(5)5；(6)0

课堂练习 2.2

在一个单处理机的系统中，OS的进程有运行、就绪、阻塞三个基本状态。假如某时刻该系统中有10个进程并发执行，在略去调度程序所占用时间情况下试问：

这时刻系统中处于运行态的进程数最多有几个？最少有几个？

这时刻系统中处于就绪态的进程数最多有几个？最少有几个？

这时刻系统中处于阻塞态的进程数最多有几个？最少有几个？

2.4 进程的控制

为了保护操作系统不被非操作系统软件破坏，将处理机的执行状态分为两种：

系统态(核心态、管态)：有特权，能执行所有指令，能访问所有寄存器和存储区。当运行在cpu上的软件为操作系统时，处理机的态处于核心态。

用户态(目态、算态)：无特权，只能执行规定指令，只能访问指定的寄存器和存储区。当运行在cpu上的软件为非操作系统时，处理机的态处于用户态。

操作系统内核是系统硬件的首次延伸，通过执行各种原语操作来实现对进程的控制功能（创建、调度、通讯、撤销等）。

2.4 进程的控制

【原语(Primitive)】：由若干条机器指令构成的并用以完成特定功能的一段程序，而且这段程序在执行期间不允许中断。原语又称为“原子操作(Atomic Operation)”过程，作为一个整体而不可分割——要么全都完成，要么全都不做。

内核中包含的进程控制的原语主要有：

1）进程创建原语

2）进程撤消原语

3）阻塞原语、唤醒原语

4）挂起原语、激活(解挂)原语

2·4 进程的控制

1、进程的创建

1）进程何时创建?

引起创建进程的事件：

用户登录、作业调度、提供服务、应用请求

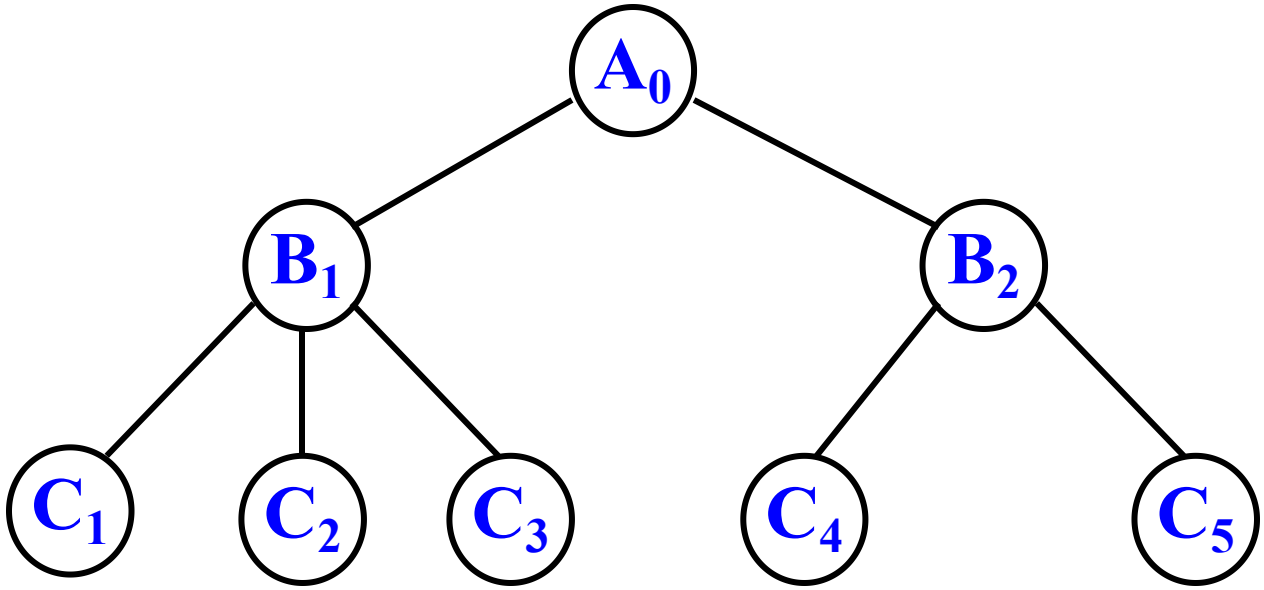
- 用户登录时， 由OS为合法终端创建一个进程
- 调度到某个批处理作业时，由作业调度程序创建
- 运行程序请求提供服务(如：打印文件)，由OS创建
- 运行中进程因自己的需要，由它自己创建子进程

2·4 进程的控制

2）进程图

允许进程创建子进程，子进程还可以创建自己的子进程，从而形成树型的进程家族。

进程图是用于描述进程家族关系的有向图，子进程可以继承父进程所拥有的资源。



2·4 进程的控制

【树型结构系统的优点】：

☞资源分配严格。祖先拥有进程家族的所有资源，子进程可在祖先进程所拥有的资源中进行分配、使用与归还。

☞进程控制灵活。可根据需要给予进程不同的控制权力，而且可根据需要创建多个子进程并行工作，协同完成任务。

☞进程层次清晰，关系明确。

3）进程如何创建？

进程的建立有两种方式：

☞系统生成是就建立起一些系统进程。主要用于创建常驻内存的系统进程；

☞经创建原语产生进程。主要用于创建非常驻的系统进程和用户进程。

2·4 进程的控制

4）进程的创建过程

一旦发现了要求创建新进程的事件，OS便调用创建原语，按以下过程创建新进程。

- 分配一个维一的进程标识符pid,索取一个空白PCB数据结构；
- 为新进程的程序和数据分配内存空间
- 初始化进程控制块

  初始化标识符信息(填入)、处理机的状态信息(指令指针，栈指针)和控制信息(状态,优先级...)
- 设置相应的链接。如：把新进程加到就绪队列的链表中

2·4 进程的控制

2、进程的中止（撤销）

1）进程何时中止?

- ☞ **正常结束。**批处理系统中,进程已运行完成遇到 *Halt* 指令；分时系统中，用户退出登录
- ☞ **异常结束。**

本进程发生出错和故障事件

存储区越界、保护性错(如:写只读文件)、特权指令错、非法指令(如:程序错转到数据区)、算术运算错、运行超时、等待超过时、I/O 失败、

- ☞ **外界干预。**

操作系统干预、父进程请求、父进程终止

2·4 进程的控制

2）进程的终止过程

一旦发生终止进程的事件，OS便调用撤消原语，按以下过程终止该进程。

- 从PCB中读取进程的状态；
- 若进程处于执行态，应立即终止该进程的执行，并置调度标志为真（以便该进程终止后系统重新进行调度，将处理机分配给新选择的进程）
- 若有子孙进程则将其全部终止，以防它们失控
- 将该进程所占有的全部资源还给父进程或系统
- 将该进程的PCB从所在队列中移出。

2·4 进程的控制

3、进程的状态的转换

1）进程的阻塞

处于运行状态的进程，在其运行过程中期待某一事件发生（如：请求系统服务、等待键盘输入、等待数据传输完成、等待其它进程发送消息）当被等待的事件未发生时，由进程调用阻塞原语(block)，将自己阻塞。

阻塞原语使处于运行态的进程停止运行，将运行现场保存在其PCB的CPU现场保护区，然后将PCB中的现行状态由运行态变为阻塞态，并将该进程插入到相应事件的阻塞队列中。最后，转进程调度程序重新调度，将处理机分配给一个就绪进程，按新进程PCB中的处理机状态设置CPU环境，使它投入运行。

2·4 进程的控制

2）进程的唤醒

当被阻塞进程期待的事件到来时，由中断处理进程或其它产生该事件的进程调用唤醒原语(Wakeup)，将期待该事件的进程唤醒。

唤醒原语执行时，将被阻塞进程从相应等队列中移出，并将其 PCB中的现行状态由阻塞改为就绪态，然后将该进程插入就绪队列中。

若事件是等待 I/O 完成，则由硬件提出中断请求，CPU响应中断，暂停当前进程的执行，转去中断处理。检查有无等待该 I/O完成的进程。若有，则将它唤醒。然后结束中断处理。返回被中断进程或重新调度。

若事件是等待某进程发一个信息，则由发送进程把该等待进程唤醒。

2·4 进程的控制

3）进程的挂起

当进程请求将自己挂起或父进程请求将子进程挂起时，调用挂起原语(suspend)，将指定进程挂起。

执行过程：检查要挂起进程的状态，若处于活动就绪态就将其改为静止就绪态，对于活动阻塞态的进程则将其改为静止阻塞态。如果被挂起的进程正在执行则还要转到调度程序重新调度。

4）进程的激活：

要激活指定进程，调用激活原语(active)将它激活。

执行过程：将要激活的进程调入内存，并检查它的状态，若是静止就绪态则将其改为活动就绪态,若为静止阻塞态就将其改为活动阻塞态。如果采用的是抢占调度策略,被激活的进程优先级高则引起重新调度。

课堂练习3

从静态角度看，进程由 A 、 B 和 C 三部分组成，用户可通过 D 建立和撤消进程，通常用户进程被建立后， E 。

A：(1)JCB；　(2)DCB；　(3)PCB (4)PMT。

B: (1)程序段；　(2)文件体　(3)I/O (4)子程序。

C：(1)文件描述块(2)数据空间(3)EOF (4)I/O缓冲区。

D：(1)函数调用；　(2)宏指令；

(3)系统调用；　(4)过程调用。

E：(1)便一直存在于系统中，直到被操作人员撤消；

(2)随着作业运行正常或不正常结束而撤消；

(3)随着时间片轮转而撤消与建立；

(4)随着进程的阻塞或唤醒而撤消与建立。

课堂练习 4

正在执行的进程由于其时间片完而被暂停执行，此时进程应从运行态变为 A 状态；处于静止阻塞状态的进程，在进程等待的事件出现后，应转变为 B 状态；若进程正处于运行态时，应终端的请求而暂停下来以便研究其运行情况(执行挂起进程原语)，这时进程应转变为 C 状态，若进程已处于阻塞状态，则此时应转变为 D 状态，若进程已处于就绪状态，则此时应转变为 E 状态；执行解除挂起进程原语后，如挂起进程处于就绪状态，则应转变为 F 态，如处于阻塞状态，则应转变为 G 态；一个进程刚被创建时，它的初始状态为 H 。

A—H：(1)静止阻塞；(2)活动阻塞；(3)静止就绪；(4)活动就绪；(5)执行。

2.5 进程同步

2.5.1 进程同步的基本概念

进程间普遍存在两种形式的制约关系：

1、间接制约（互斥）：并发进程由于共享了某种特殊的资源而必须互相等待。

2、直接制约（同步）：功能上相关的程序在某些特殊的时刻必须互相通信，并根据通信的结果决定继续还是暂停。

进程间的制约关系普遍存在！

临界资源和临界区

特殊的资源：临界资源（Critical Resource），一次只能为一个进程使用的资源。大部分硬件设备和公共变量都属于临界资源。

临界区（Critical Segment）：程序中涉及临界资源使用的代码片段。

操作系统进程同步机制

在进程并发执行时，进程间制约关系（互斥和同步）必须得到满足，否则程序的执行不正确（无法满足用户的预期）。

如何让满足?

必须由操作系统来实现进程间制约关系。

操作系统提供进程同步机制：

① 数据（初值，意义）。用于描述进程并发过程中需要进行控制的对象。数据有非负的初值。

② 定义在数据上的原语操作。对数据的操作由用户在程序中调用原语来实现。不同的操作由不同的原语实现。

信号量机制

在多道环境下，并发进程表现为"走一走停一停"的特点。何时停？何时走？

在操作系统提供的同步机制下，所有的进程间的制约关系得以实现。

信号量机制是操作系统为实现进程间制约关系而提供的一种进程同步机制，类比于现实生活中红绿灯控制交通的场景：

道路	计算机中的资源
共享道路的车和人	发进程
信号灯的颜色（红、绿、黄）	信号量中数据的值
信号灯的动作（切换）	定义在信号量上的原语操作
规则（红灯停，绿灯行）	原语的定义：数据的值的范围的判断）

记录型信号量

信号量机制有整形信号量、记录型信号量，and信号量和信号量集。

记录型信号量机制：

① 数据，结构体变量。　　数据有非负的初值。

两个分量：1、int value 用于描述进程并发过程中需要进行控制的对象。

2、PCB *list等待队列。记录了所有由于信号量value的值不合适而必须等待的进程。

② 原语。P操作和V操作。定义在数据上的原语操作。
对数据的操作由用户在程序中调用原语来实现。

信号量的定义

semaphore S;

用S.value和S.list来引用记录型信号量的两个分量。

定义在信号量S上的原语是两个操作系统功能函数。

P操作（wait操作）

```
wait(S)
  {
     S.value  =S.value-1;
       if S.value＜0
            block(S.List)
     }
```

```
P(S)
  {
     S.value  =S.value-1;
       if S.value＜0
            block(S,List)
     }
```

P操作的直观含义

```
P(S)
 {
    S.value  =S.value-1;
     if S.value＜0
          block(S.List)
    }
```

信号量S的值value:

表示当时系统中某类临界资源的可用个数。

P(s)这一操作的直观含义是什么

V操作（signal操作）

```
signal(S)
  {
    S.value  =S.value+1;
       if S.value<=0
            wakeup(S.List)
    }
```

```
V(S)
  {
    S.value  =S.value+1;
       if S.value<=0
            wakeup(S.List)
    }
```

V操作（signal操作）

```
V(S)
 {
   S.value  =S.value+1;
     if S.value<=0
          wakeup(S.List)
   }
```

信号量S的值value:

表示当时系统中某类临界资源的可用个数。

V(s)这一操作的直观含义是什么

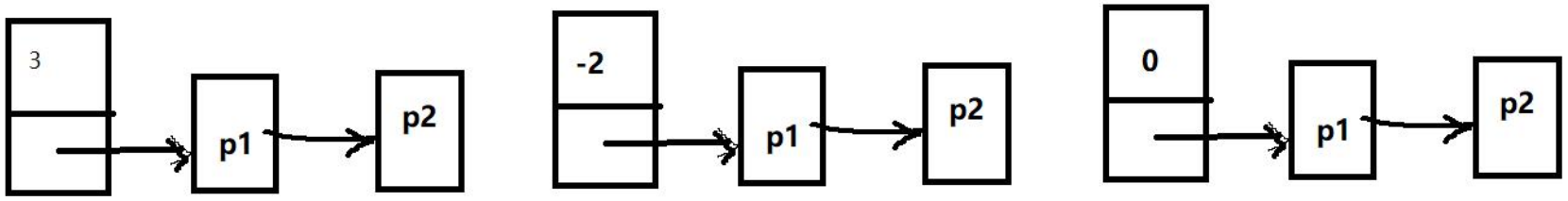
信号量原语

```
P(S)
  {
     S.value  =S.value-1;
       if S.value＜0
            block(S.List)
     }
```

```
V(S)
  {
     S.value  =S.value+1;
        if S.value<=0
             wakeup(S.List)
      }
```

信号量的取值有啥规律?

关于信号量的取值，下列哪一种情况的取值是可能的?



根据信号量上P,V原语的语义，信号量的取值有如下规律:

S.value>=0时，S.list = NULL

S.value<0时，挂在S.list上处于等待状态的进程的个数为 abs(S.value)个

用信号量实现进程互斥

例：N个进程并发执行，共享一个临界资源。试用信号量实现N个进程互斥。

解题步骤：

1、分析制约关系，明确要控制的对象。（信号量的个数，每个信号量的意义，初值）

2、确定控制细节：

- when:控制的时刻（关键代码点）
- who:控制的对象（哪个信号量）
- how:控制方式（哪个原语）

3、伪代码实现（书写规范）

N个进程并发执行，共享一个临界资源。试用信号量实现N个进程互斥。

1、分析制约关系，明确要控制的对象。（信号量的个数，每个信号量的意义，初值）

2、确定控制细节：

when:控制的时刻（关键代码点）

who:控制的对象（哪个信号量）

how:控制方式（哪个原语）

3、伪代码实现（书写规范）

1、N个进程功能上无关，并发时共享一个临界资源。单纯的互斥。

要控制的对象：临界资源。设置一个信号量S来描述该CR。S.value表示任意时刻系统中该资源的可用个数。S.list上挂着为因为未申请到该临界资源而等待的进程。在初始条件下S.value=1，S.list=NULL

2、

when:对于每一个进程，需要控制的时刻有两个：进入临界区之前，离开临界区之后

who:系统中的该CR,用信号量S描述

How:进入临界区前，进行资源申请(用原语P申请)

离开临界区后，进行资源释放（用原语V释放）

伪代码实现

1、定义信号量，赋初值和意义

semaphore S ;　s.value = 1; s.list = NULL;/*S用来描述要控制的CR，S.value表示任一时刻系统中该CR的可用个数

2、主程序

main()

cobegin(parbegin)

p_1;

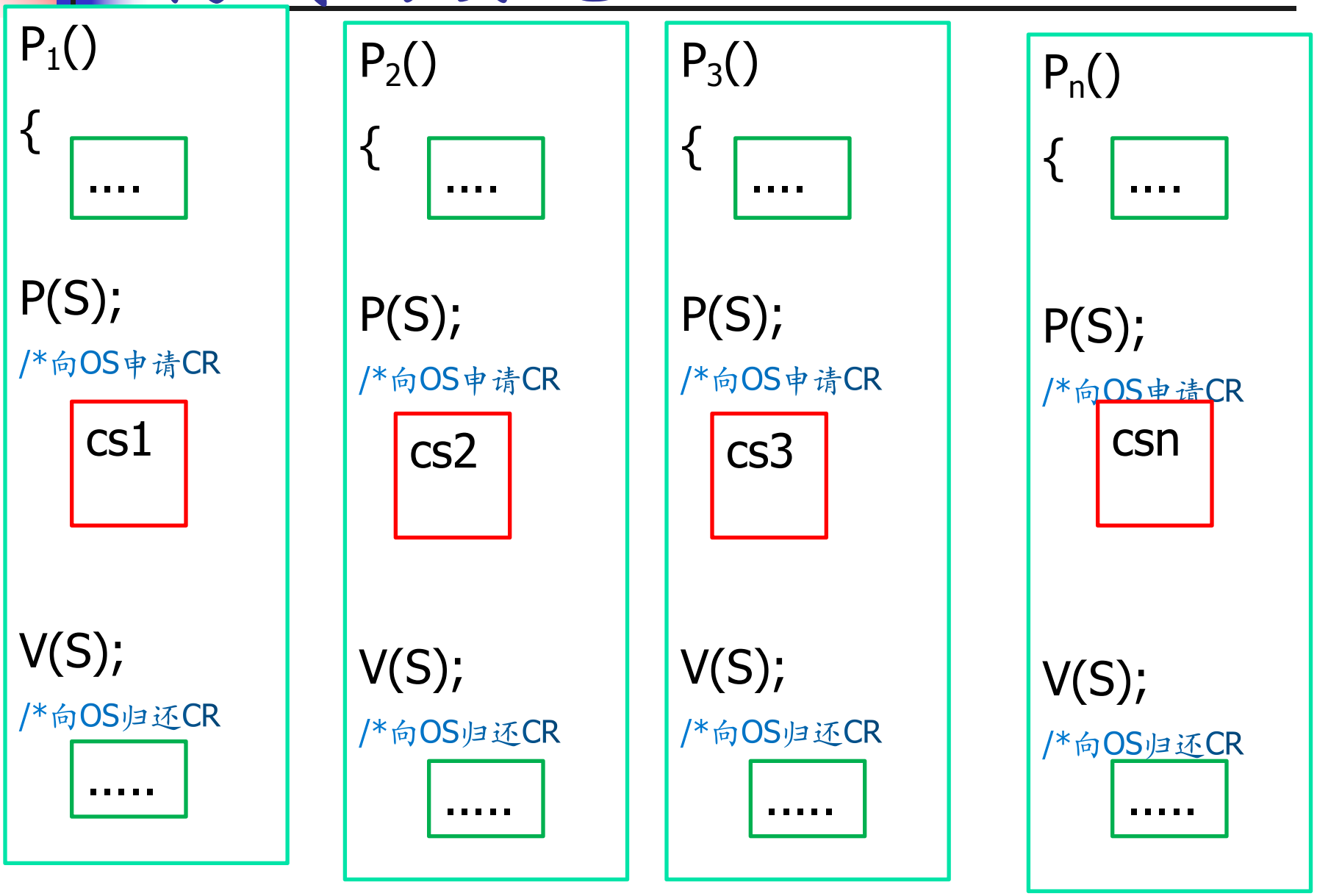
p_2;

....

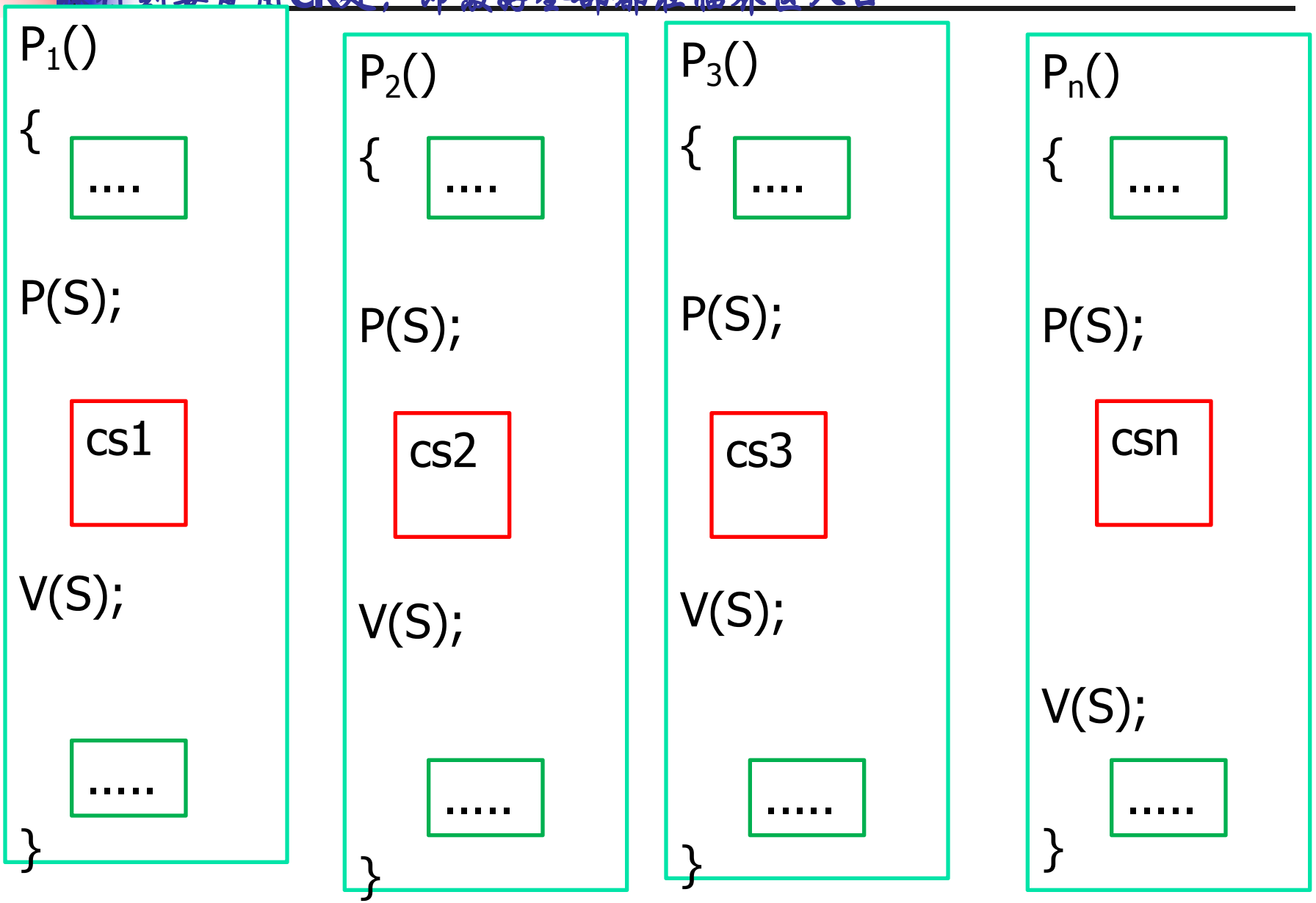
p_n

coend(parend)

伪代码实现


P1()
{
....
P(S);
/*向OS申请CR
cs1
V(S);
/*向OS归还CR
.....
P2()
{
....
P(S);
/*向OS申请CR
cs2
V(S);
/*向OS归还CR
.....
P3()
{
....
P(S);
/*向OS申请CR
cs3
V(S);
/*向OS归还CR
.....
Pn()
{
....
P(S);
/*向OS申请CR
csn
V(S);
/*向OS归还CR
.....

验证：n个进程按照P1,p2,....pn的顺序排在就绪队列上，所有进程都恰好执行到要使用CR处，即做好全部都在临界区入口

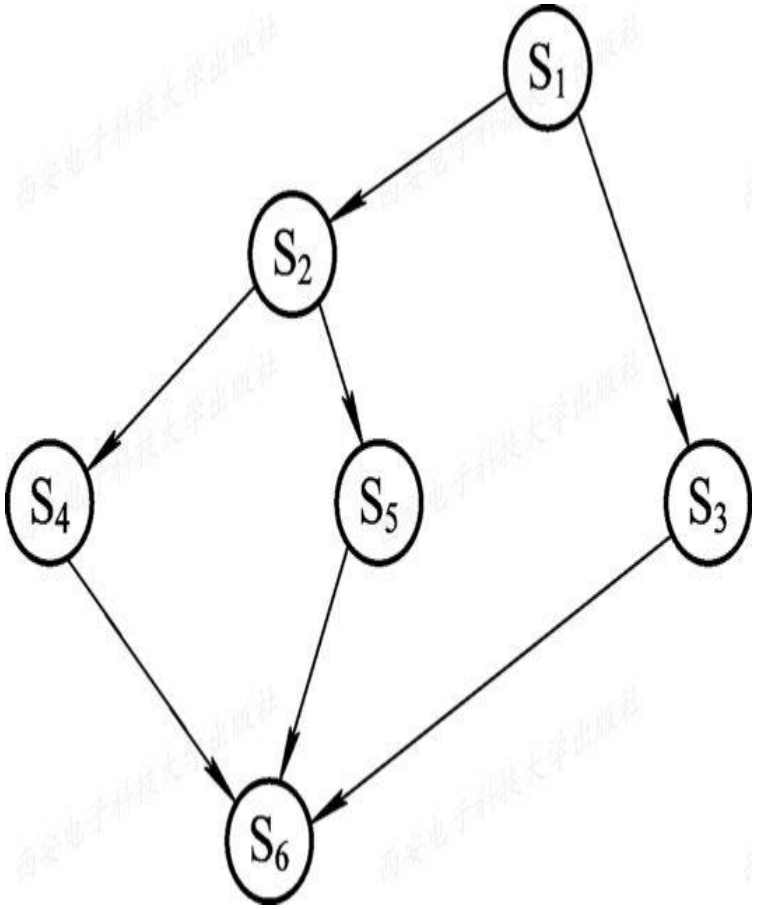


用信号量实现进程同步

1、前驱图

指一个有向无循环图，可记为DAG(Directed Acyclic Graph)，它用于描述进程之间执行的先后顺序。

图中的每个结点可用来表示一个进程或程序段，乃至一条语句，结点间的有向边则表示两个结点之间存在的偏序(Partial Order)或前趋关系



概念：（直接）前驱，

（直接）后继

$P_i \rightarrow P_j$:一对直接前驱后继关系

用信号量实现进程同步

6个并发进程的执行顺序如前驱图所示，用信号量实现同步关系。

解题步骤：

1、分析制约关系，明确要控制的对象。（信号量的个数，每个信号量的意义，初值）

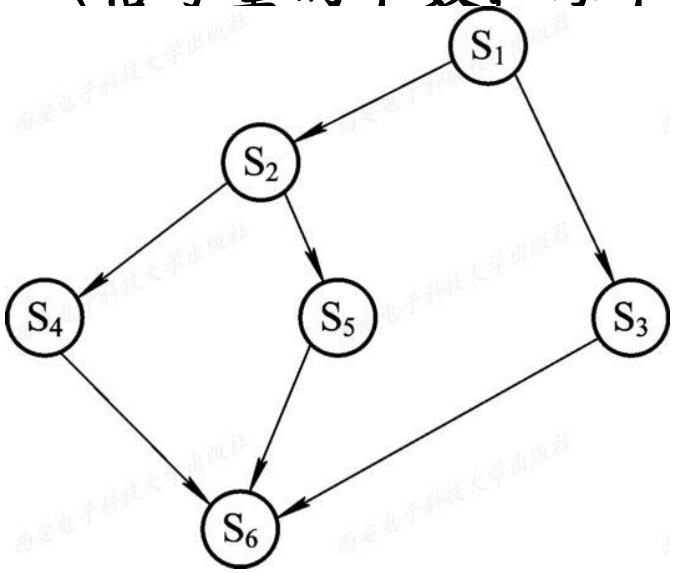
2、确定控制细节：

when:控制的时刻（关键代码点）

who:控制的对象（哪个信号量）

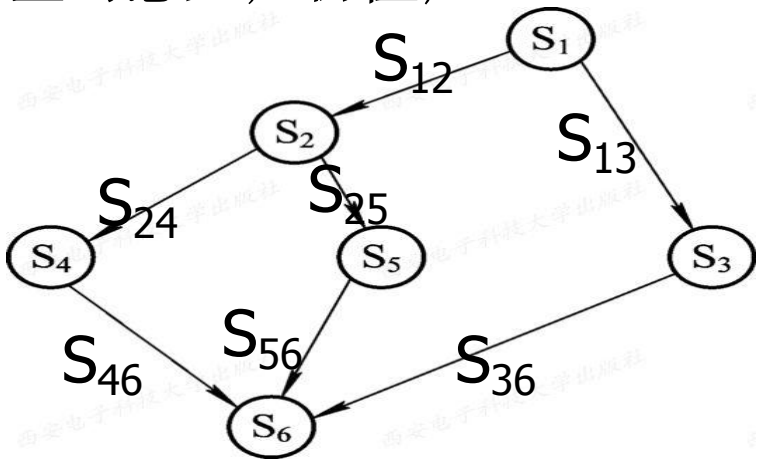
how:控制方式（哪个原语）

3、伪代码实现（书写规范）



6个并发进程的执行顺序如前驱图所示，用信号量实现同步关系。

1、分析制约关系，明确要控制的对象。（信号量的个数，每个信号量的意义，初值）



2、确定控制细节：

when:控制的时刻（关键代码点）

who:控制的对象（哪个信号量）

how:控制方式（哪个原语）

3、伪代码实现（书写规范）

1、6个进程功能上相关，并发时共必须按照前驱图规定的顺序执行，否则执行结果不正确。

要控制的对象：每一对直接前驱后继关系：

分别设置信号量$S_{12},S_{13},S_{24},S_{25},S_{46},S_{56},S_{36}$=?

2、

when:对于每一个进程，需要控制的时刻有两个：进程开始执行之前，进程执行结束之后

who:每一对直接前驱后继关系

How:开始执行前，检查前驱，逐一询问（何种原语操作）前驱执行完没有，yes,开始执行，No,挂起，直到前驱完成后将其唤醒；没有前驱，开始执行

P操作!

进程结束之后，检查后继，逐一通知（何种原语操作）后继有后继等待自己完成，唤醒，

V操作!

伪代码实现

1、定义信号量，赋初值和意义

semaphore $S_{12},S_{13},S_{24},S_{25},S_{46},S_{56},S_{36}=0$; ;/*$S_{ij}$表示进程$P_i$和$P_j$之间的直接前驱后继关系

2、主程序

main()

cobegin(parbegin)

p_1;

p_2;

....

p6

coend(parend)

```
P1()
{
  ...
V(S12);
/*通知P2
V(S13);
/*通知P3
}
```

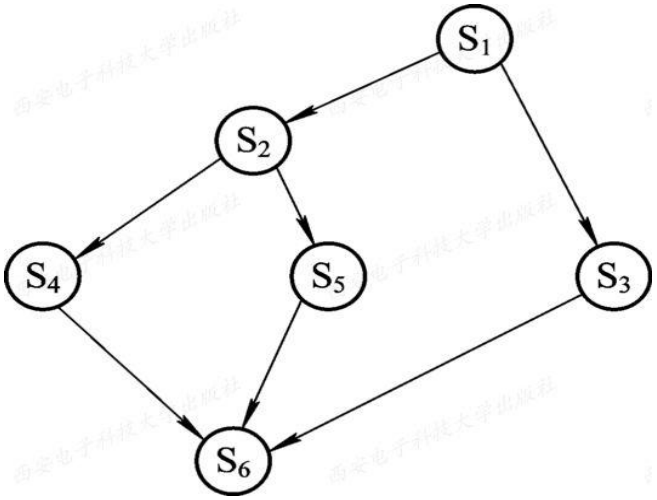
```
P2()
{
P(S12);
/*询问p1
  ...
 V(S24);
/*通知P4
V(S25);
/*通知P5
}
```

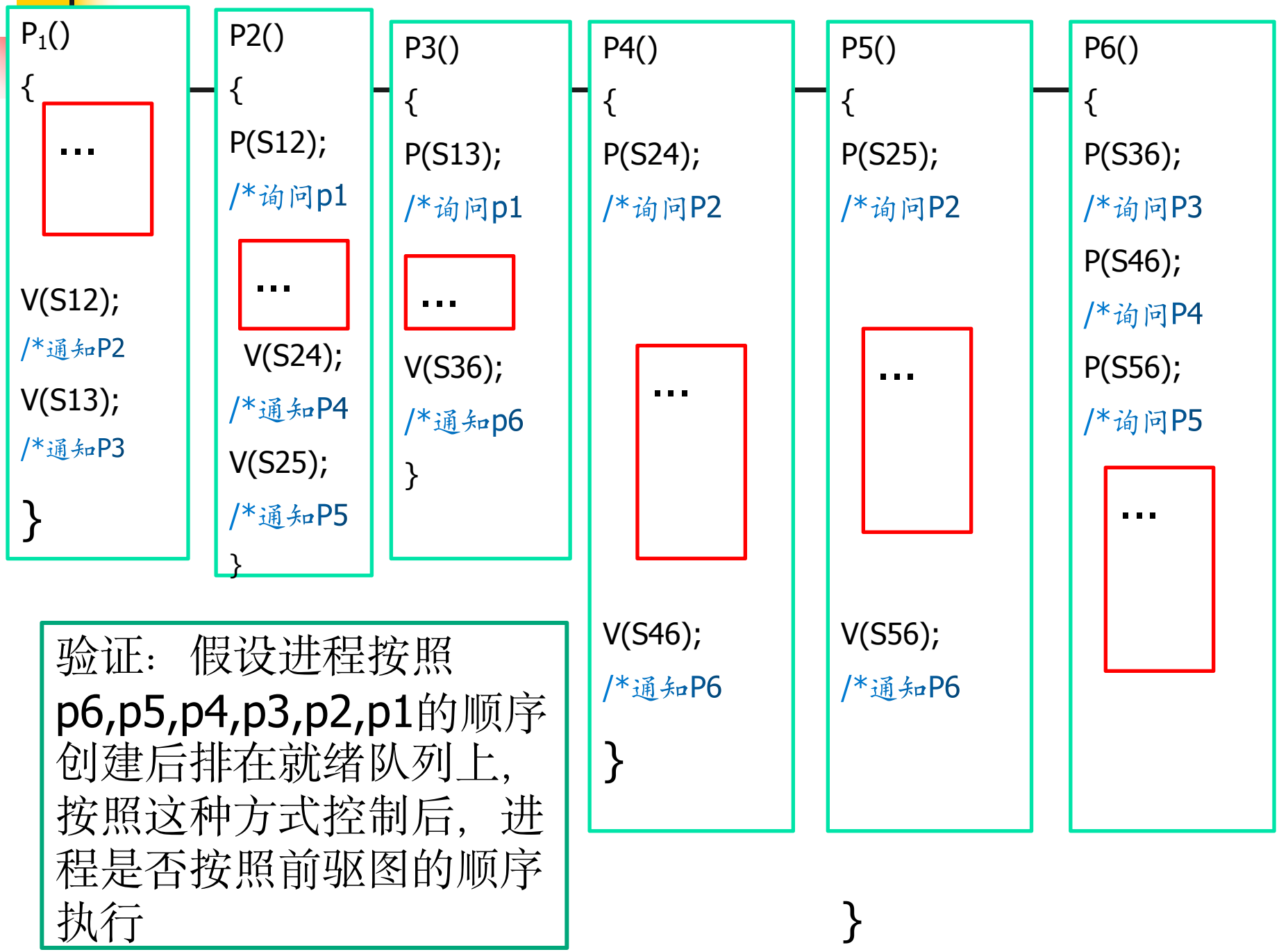
```
P3()
{
P(S13);
/*询问p1
  ...
V(S36);
/*通知p6
}
```

```
P4()
{
P(S24);
/*询问P2
  ...
V(S46);
/*通知P6
}
```

```
P5()
{
P(S25);
/*询问P2
  ...
V(S56);
/*通知P6
}
```

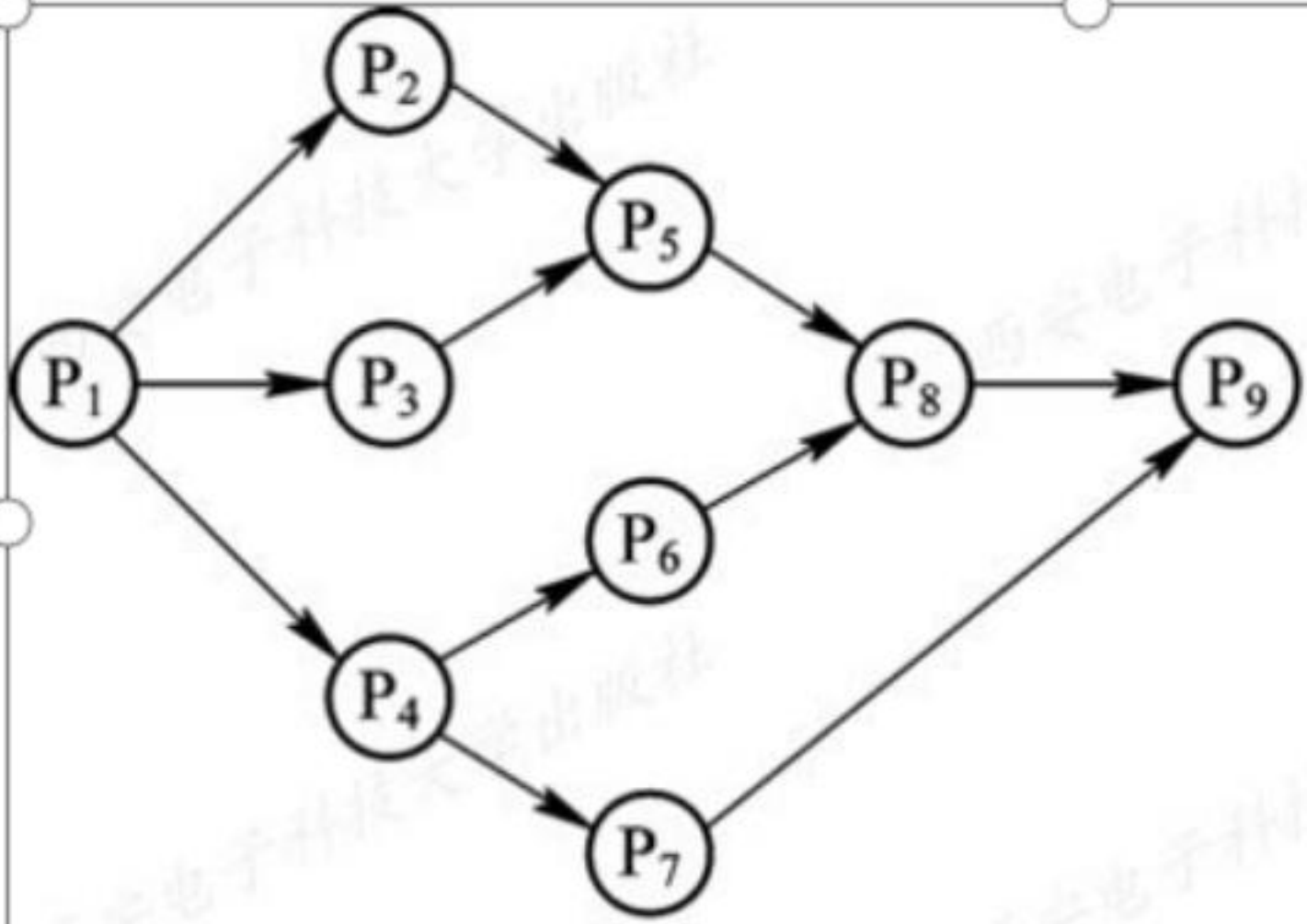
```
P6()
{
P(S36);
/*询问P3
P(S46);
/*询问P4
P(S56);
/*询问P5
  ...
}
```





验证：假设进程按照p6,p5,p4,p3,p2,p1的顺序创建后排在就绪队列上，按照这种方式控制后，进程是否按照前驱图的顺序执行

练习：用信号量机制控制进程间同步



(a) 具有九个结点的前趋图

思考：

对于例题中的前驱图，是否必须7个信号量才能控制住？

扩展阅读：《用信号量实现进程同步的一题四解方法》

经典的进程同步问题

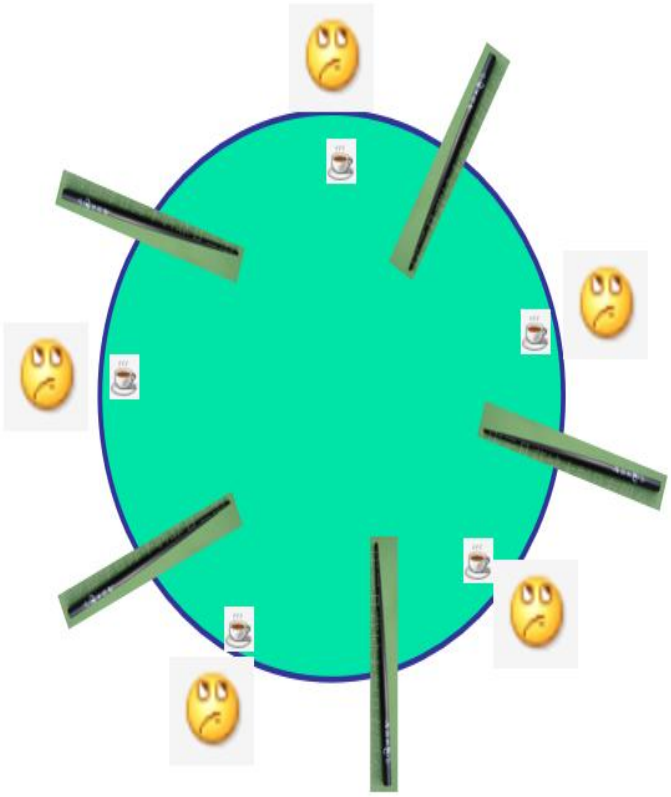
在多道程序环境下，进程同步问题十分重要，也是相当有趣的问题，因而吸引了不少学者对它进行研究，产生了经典的进程同步问题。通过对这些问题的研究和学习，可以帮助我们更好地理解进程同步的概念及实现方法。

1、哲学家吃饭问题

2、生产者消费者问题（PC问题）

3、读者写者问题（RW问题）

1、哲学家吃饭问题



1、问题描述

有五个哲学家，他们的生活方式是交替地进行思考和进餐，哲学家们共用一张圆桌，分别坐在周围的五张椅子上，在圆桌上有五个碗和五支筷子，平时哲学家进行思考，饥饿时便试图取其左、右最靠近他的筷子，只有在他拿到两支筷子时才能进餐，该哲学家进餐完毕后，放下左右两只筷子又继续思考。

2、约束条件

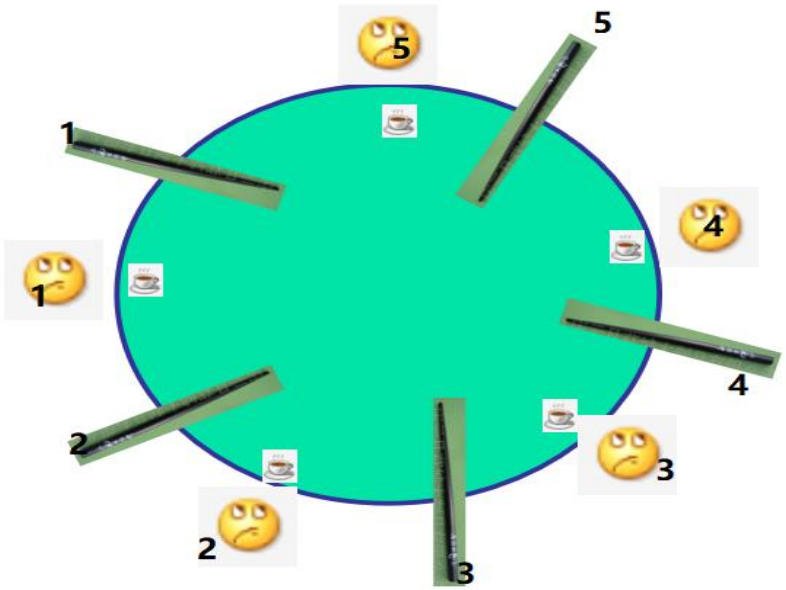
(1)只有拿到两只筷子时，哲学家才能吃饭。

(2)如果筷子已被别人拿走，则必须等别人吃完之后才能拿到筷子。

(3)任一哲学家在自己未拿到两只筷子吃完饭前，不会放下手中已经拿到的筷子。

试用信号量实现5个哲学家吃饭问题。

1、分析制约关系，明确要控制



2、确定控制细节：

when:控制的时刻（关键代码点）

who:控制的对象（哪个信号量）

how:控制方式（哪个原语）

3、伪代码实现（书写规范）

1、5个哲学家的日常工作（思考和吃饭交替进行）并发执行。5个哲学家的思考过程任意并发，吃饭时因为守资源限制（5只筷子）必须互相制约，每一只筷子都是临界资源，必须互斥地使用。

要控制的对象：每一只筷子：

分别设置信号量$S_1,S_2,S_3,S_4,S_5=1$

2、

when:对于每一个哲学家，需要控制的时刻有两个：开始吃饭之前，吃饭结束之后

who:每一只筷子的互斥使用

How:开始吃饭之前，看左手边筷子空闲否，yes,拿起，No,等待，（ ）直到左手边筷子被放回原处，将其唤醒；

P操作

吃饭结束之后，放下左手边筷子，放下右手边筷子

V操作

伪代码实现

1、定义信号量，赋初值和意义

semaphore S_1,S_2,S_3,S_4,S_5=**1**；；/*S_i表示第i只筷子，value表示该只筷子的可用数量

2、主程序

main()

cobegin(parbegin)

p_1;

p_2;

....

p5;

coend(parend)

$P_1()$

{repeat

thingking...;

P(S1);

/*左手边筷子空否

P(S2);

/*右手边筷子空否

eating...;

V（S1）；

/*放下左手边筷子

V(S2);

/*放右左手边筷子

until false}

P2()

{repeat

thingking...;

P(S2);

/*左手边筷子空否

P(S3);

/*右手边筷子空否

eating...;

V（S2）；

/*放下左手边筷子

V(S3);

/*放右左手边筷子

until false}

P3()

{repeat

thingking...;

P(S3);

/*左手边筷子空否

P(S4);

/*右手边筷子空否

eating...;

V（S3）；

/*放下左手边筷子

V(S4);

/*放右左手边筷子

until false}

P4()

{repeat

thingking...;

P(S4);

/*左手边筷子空否

P(S5);

/*右手边筷子空否

eating...;

V（S4）；

/*放下左手边筷子

V(S5);

/*放右左手边筷子

until false}

P5()

{repeat

hingking...;

P(S5);

/*左手边筷子空否

P(S1);

/*右手边筷子空否

eating...;

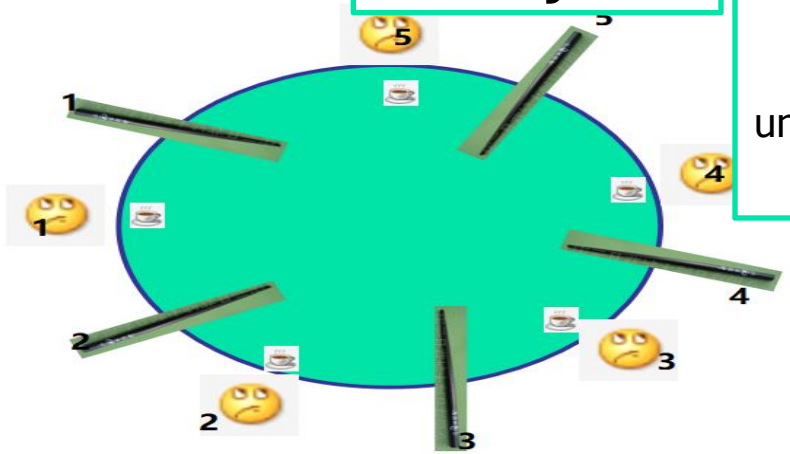
V（S5）；

/*放下左手边筷子

V(S1);

/*放右左手边筷子

until false}



思考：

当五个哲学家同时饥饿时，会发生什么现象？

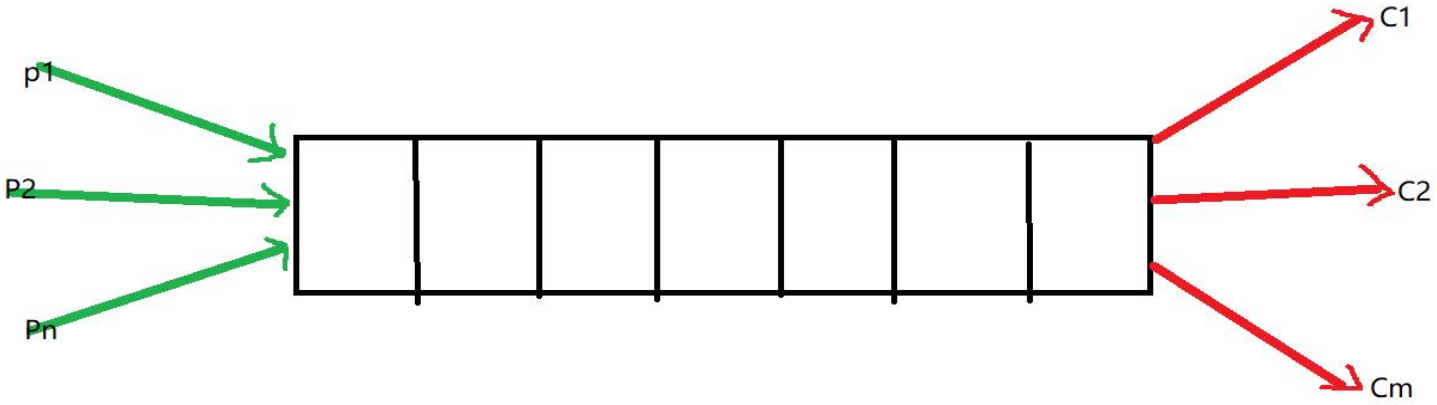
死锁！

有什么办法解决？

生产者消费者问题

1、 问题描述：

多个（n+m）进程并发执行，共享一个有界（大小为N）缓冲区，其中一类进程生产产品后将产品放入缓冲区（生产者），一类进程从缓冲区中取出产品，消耗掉（消费者）。



约束条件：

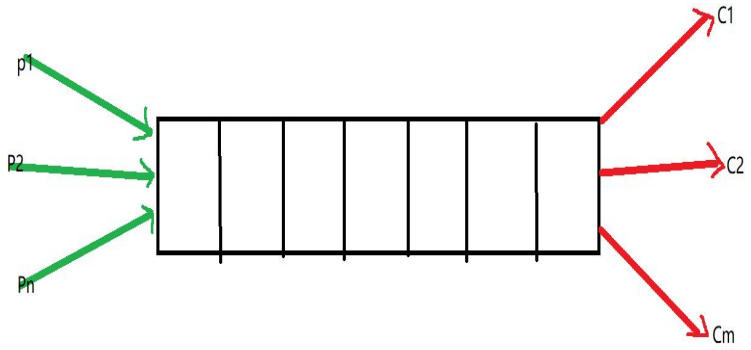
1、整个缓冲区为临界资源，一次只有一个进程访问该临界资源

2、生产者当缓冲区中有空位时一次向缓冲区中放入一个产品，否则等待

3、消费者在缓冲区中有产品时一次从缓冲区中取出一个产品，否则等待

试用信号量实现生产者消费者问题。

1、分析制约关系，明确要控制的对象。（信号量的个数，每个信号量的意义，初值）



2、确定控制细节：

when:控制的时刻（关键代码点）

who:控制的对象（哪个信号量）

how:控制方式（哪个原语）

3、伪代码实现（书写规范）

1、在PC问题中有如下制约关系：

①互斥：有界缓冲区是一个CR，所有进程（pp,cc,pc）都必须互斥地使用。为了实现互斥，设置一个互斥信号量S,用于描述当前有界缓冲能否使用。s.value=1

②同步：生产者：最关注的缓冲区中"空格子"的数量，有空格子就可以放产品否则等待（等消费者拿走产品后被唤醒）

消费者：最关注缓冲区中"满格子"（产品）数量，有产品才可以拿，否则等待（等生产者放入产品后被唤醒）

要控制的对象：缓冲区中空格子和满格子

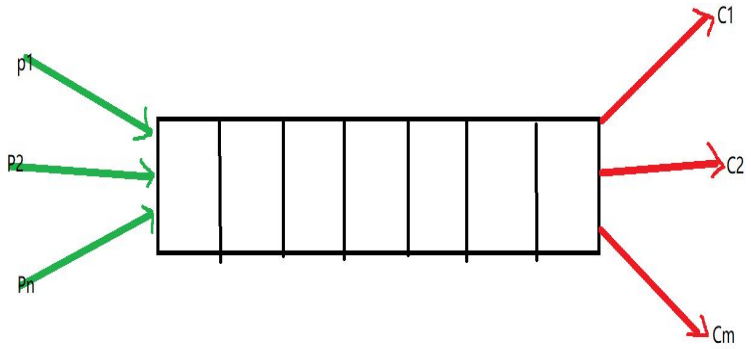
分别设置信号量empty和full.

初始条件下，empty.value= N,full.value = 0

任何时候empty.value+full.value = N

试用信号量实现生产者消费者问题。

1、分析制约关系，明确要控制的对象。（信号量的个数，每个信号量的意义，初值）



2、确定控制细节：

when:控制的时刻（关键代码点）

who:控制的对象（哪个信号量）

how:控制方式（哪个原语）

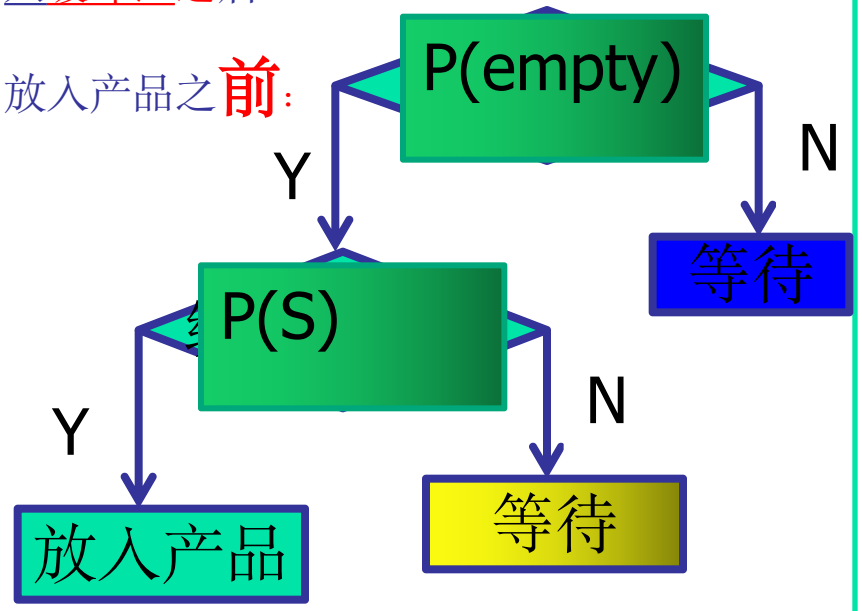
3、伪代码实现（书写规范）

semaphore s = 1,empty =N,full = 0

2、确定控制细节：

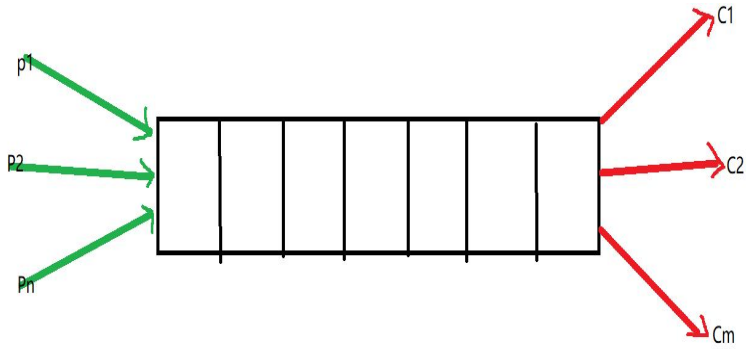
when:对于每一个生产者，需要控制的时刻有两个：向缓冲区放产品之前，将产品放入缓冲区之后

放入产品之前：



试用信号量实现生产者消费者问题。

1、分析制约关系，明确要控制的对象。（信号量的个数，每个信号量的意义，初值）



2、确定控制细节：

when:控制的时刻（关键代码点）

who:控制的对象（哪个信号量）

how:控制方式（哪个原语）

3、伪代码实现（书写规范）

semaphore s = 1,empty =N,full = 0

2、确定控制细节：

when:对于每一个**生产者**，需要控制的时刻有两个：向缓冲区放产品之前，将产品放入缓冲区之后

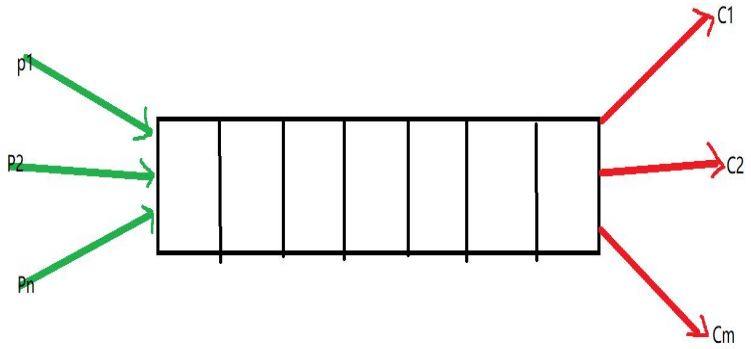
放入产品之**后**：

释放有界缓冲的控制权 → V(S)

通知**消费者**：放入一个产品 → V(full)

试用信号量实现生产者消费者问题。

1、分析制约关系，明确要控制的对象。（信号量的个数，每个信号量的意义，初值）



2、确定控制细节：

when:控制的时刻（关键代码点）

who:控制的对象（哪个信号量）

how:控制方式（哪个原语）

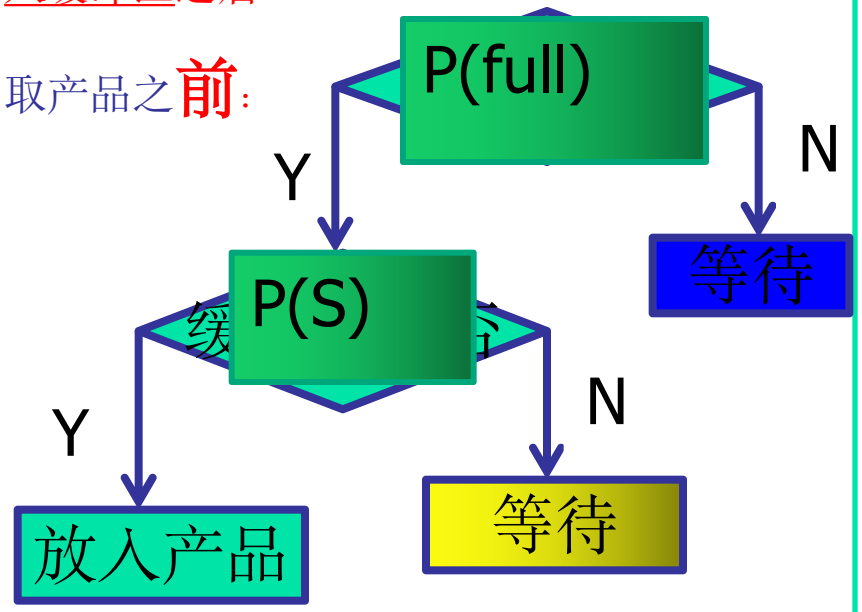
3、伪代码实现（书写规范）

semaphore s = 1,empty =N,full = 0

2、确定控制细节：

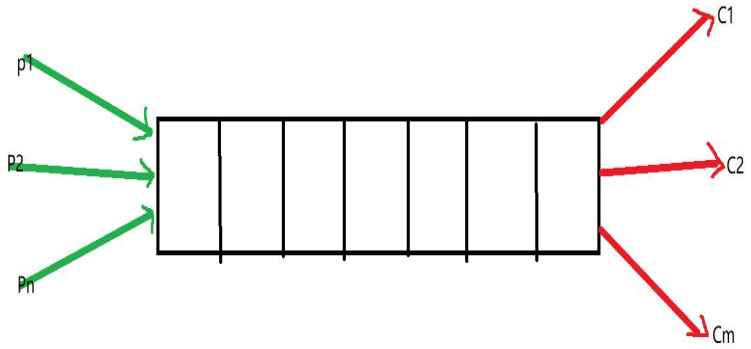
when:对于每一个**消费者**，需要控制的时刻有两个：从**缓冲区取产品之前**，将产品放入缓冲区之后

取产品之**前**：



试用信号量实现生产者消费者问题。

1、分析制约关系，明确要控制的对象。（信号量的个数，每个信号量的意义，初值）



2、确定控制细节：

when:控制的时刻（关键代码点）

who:控制的对象（哪个信号量）

how:控制方式（哪个原语）

3、伪代码实现（书写规范）

semaphore s = 1,empty =N,full = 0

2、确定控制细节：

when:对于每一个**消费者**，需要控制的时刻有两个：从**缓冲区取产品之前**，将产品取出缓冲区之后

取产品之**后**：

释放有界缓冲的控制权 → V(S)

通知**生产者**：产生一个空格 → V(empty)

伪代码实现

1、定义信号量，赋初值和意义

semaphore s.value=**1** ;/*用于控制所有进程互斥地使用缓冲区

empty.value = N;/*缓冲区中空格数

full.value = 0/*缓冲区中产品数

2、主程序

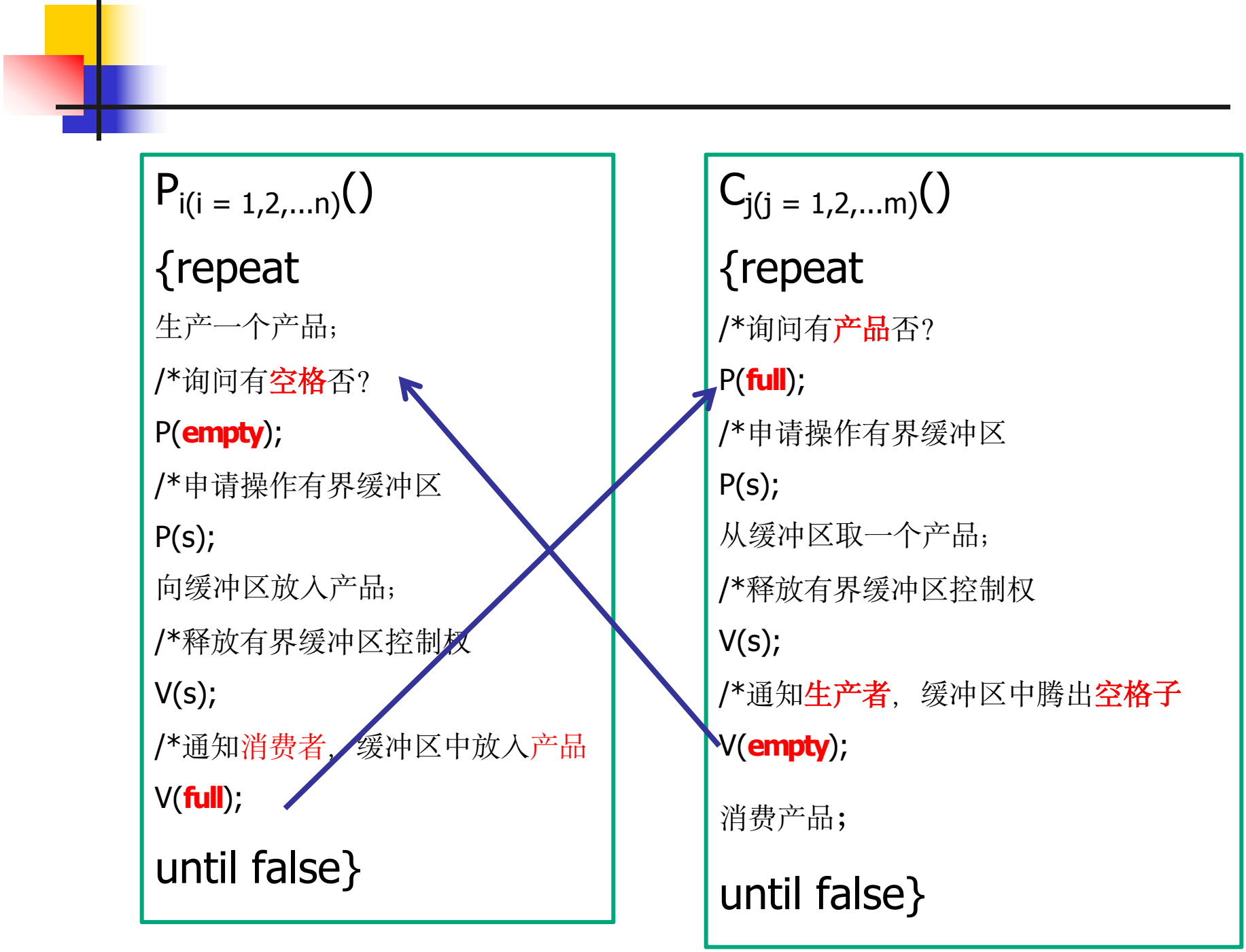
main()

cobegin(parbegin)

pi(i=1,2,...n);

Cj(j = 1,2,...m);

coend(parend)


$P_{i(i = 1,2,...n)}()$
{repeat
生产一个产品；
/*询问有空格否？
P(empty);
/*申请操作有界缓冲区
P(s);
向缓冲区放入产品；
/*释放有界缓冲区控制权
V(s);
/*通知消费者，缓冲区中放入产品
V(full);
until false}
$C_{j(j = 1,2,...m)}()$
{repeat
/*询问有产品否？
P(full);
/*申请操作有界缓冲区
P(s);
从缓冲区取一个产品；
/*释放有界缓冲区控制权
V(s);
/*通知生产者，缓冲区中腾出空格子
V(empty);
消费产品；
until false}

验证：一开始，m个消费者进程，n个生产者进程都已经创建好，生产者生产一个产品需要两个时间片。

$P_{i(i = 1,2,...n)}()$

{repeat

生产一个产品；

P(**empty**);

P(s);

向缓冲区放入产品；

V(s);

V(**full**);

until false}

$C_{j(j = 1,2,...m)}()$

{repeat

P(**full**);

P(s);

从缓冲区取一个产品；

V(s);

V(**empty**);

消费产品；

until false}

思考，生产者进程代码改成右图所示，要否？为何

```
Pi(i = 1,2,...n)()
{repeat
生产一个产品;
P(empty);

P(s);

向缓冲区放入产品;
V(s);

V(full);

until false}
```

```
Pi(i = 1,2,...n)()
{repeat
生产一个产品;
P(s);

P(empty);

向缓冲区放入产品;
V(s);

V(full);

until false}
```

思考，消费者进程代码改成右图所示，妥否？为何

$C_{j(j = 1,2,...m)}()$

{repeat

P(**full**);

P(s);

从缓冲区取一个产品;

V(s);

V(**empty**);

消费产品;

until false}

$C_{j(j = 1,2,...m)}()$

{repeat

P(s);

P(**full**);

从缓冲区取一个产品;

V(s);

V(**empty**);

消费产品;

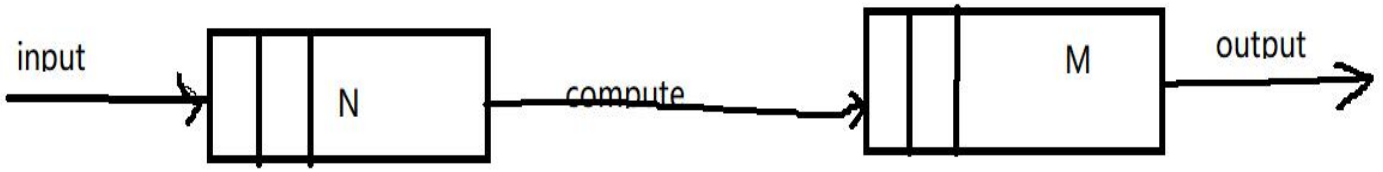
until false}

课堂练习

1、在测量控制系统中，数据采集任务把所采集的数据送入一单缓冲区;计算任务从该单缓冲区中取出数据进行计算。试写出利用信号量机制实现两者共享单缓冲区的同步算法。

2、三个进程input,compute,output共享两个有界缓冲区，大小分别为N和M.

input进程从键盘接受输入并放入缓冲区1，compute进程从缓冲区1取一个字符，转换成大写，放入缓冲区2.output进程从缓冲区2取出字符，在屏幕上显示。



3、桌上有一空盘，允许存放一只水果。爸爸可向盘中放苹果，也可向盘中放桔子，儿子专等吃盘中的桔子，女儿专等吃盘中的苹果。规定当盘空时一次只能放一只水果供吃者取用，请用P、V原语实现爸爸、儿子、女儿三个并发进程的同步。

伪代码实现

1、定义信号量，赋初值和意义

semaphore s.value=**1** ;/*用于控制所有进程互斥地使用盘子

empty.value = 1;/*盘子中空位数

apple.value = 0/*盘子中苹果数

orange.value = 0/*盘子中桔子数

2、主程序

main()

cobegin(parbegin)

father;

son;

daughter();

coend(parend)

```
f()
{repeat
处理一个水果;
p(empty);
p(s);
放入盘中;
v(s);
if 放的是苹果
  V(apple)
else
V(orange)

until false}
```

```
s()
{repeat
p(apple)
p(s)
从盘中取苹果;
V(S)
v(empty)
吃掉
until false}
```

```
d()
{repeat
p(orange)
p(s)
从盘中取桔子
V(s)
V(empty)
吃掉
until false}
```

读者写者问题

1、问题描述

一组并发进程在并发过程中共享一个数据对象DataObject（数据文件、数据库中的一张表），根据进程的功能将进程分为两类：

读者进程Readers:对数据对象执行读操作

写者进程Writters:对数据对象执行写操作

2、约束条件

写数据对象时：必须互斥，否则数据对象的值不可预料

读数据对象时：任意并发

试用信号量实现读者写者问题。

1、分析制约关系，明确要控制的对象。

1、在RW问题中有如下制约关系：

①互斥：从**写者角度**看来数据对象DO是临界资源。写者进程与任何进程（其他写者和所有读者）都是互斥地。因此，可以设置一个互斥信号量W来控制写者写者，写者与读者之间的互斥 。semaphore W.value = 1

②从**读者角度**来看，根据当前正在访问数据对象的进程的类型分三种情况：

•写者：等待 w.value<=0 and readcount = 0

•读者：直接读 readcount >0 and w.value<=0

•二者皆不是：数据对象 w.value = 1 and readcount = 0 读者，必须制造一个类似单向门的效果：

必须要有一个能将这三种情况区别开的辅助数据!

这个数据必须能实时描述**当前正在对数据对象进行读操作的读者的个数**!

int readcount = 0

readcount如何实时描述正在读DO的读者数量?

读者获得数据对象的控制权后，读之前，对readcount+1,读完之后，readcount - 1

由于多个读者并发执行，并发过程中异步地修改readcount的值。

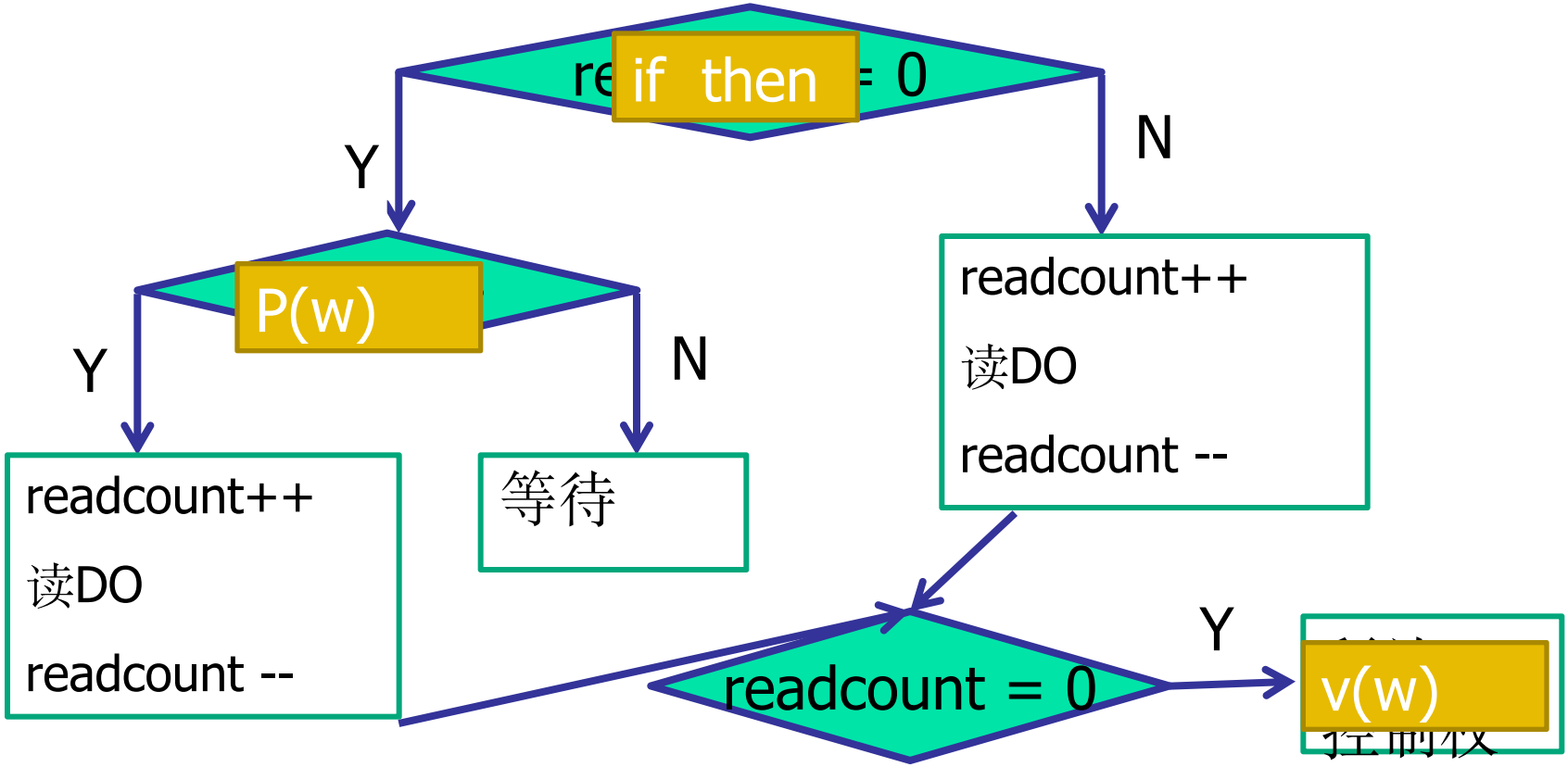
readcount是临界资源，读者们必须互斥地修改!

设置一个互斥信号量R,控制多个读者互斥地修改临界资源readcount

semaphore R.value =1

2、控制细节

1. 对于写者：在写数据对象之前，申请临界资源的控制权，写完之后，释放资源
2. 对于读者：读数据对象前，判断当前处于三种情况中的哪一种？



伪代码实现

1、定义信号量，赋初值和意义

semaphore w.value=**1** ;/*用于描述DO的控制权

int readcount = 0;/*记录当前正在数据对象上进行读操作的读者数量

semaphore R.value = 1;/*用于控制多个读者对readcount进行互斥访问

2、主程序

main()

cobegin(parbegin)

Wi(i=1,2,...n);

Rj(j = 1,2,...m);

coend(parend)

Wi(i = 1,2,...n)()

{repeat

.....

/*申请DO的控制权

P(w);

写Do;

/*释放DO的控制权

V(w);

.....

until false}

Rj(j = 1,2,...m)()

{repeat

.....

/*判断三种情况

P(R)

if readcount == 0 then P(w);

readcount ++;

V(R)

读DO;

P(R)

readcount --

if readcount == 0 then V(w);

V(R)

.....

until false}

验证：1、R1R2R3W1R4W2R5W3

2、W1R1R2R3W23R4R5

Wi(i = 1,2,...n)()

{repeat

.....

/*申请DO的控制权

P(w);

写Do;

/*释放DO的控制权

V(w);

.....

until false}

Rj(j = 1,2,...m)()

{repeat

.....

/*判断三种情况

P(R)

if readcount == 0 then P(w);

readcount ++;

V(R)

读DO;

P(R)

readcount --

if readcount == 0 then V(w);

V(R)

.....

until false}

课堂练习

(武汉理工大学2002年试题)过独木桥问题：一条小河上有一座独木桥，假设河东、河西都有人要过桥，为了保证安全，规定只要桥上无人，则允许一方的人过桥，待一方的人全部过完后，另一方的人才允许过桥。如果把每个过桥者看做一个进程，请用P、V操作实现正确管理。

伪代码实现

1、定义信号量，赋初值和意义

2、主程序

```
main()
cobegin(parbegin)
WtoE(i=1,2,...n);
EtoW(j = 1,2,...m);
coend(parend)
```

WtoEi(i = 1,2,...n)()

{repeat

.....

从西向东过桥;

.....

until false}

EtoW(j = 1,2,...m)()

{repeat

.....

从东向西过桥;

.....

until false}

2.6 线程及其管理

1、线程概念的引入

作为并发执行的进程具有二个基本的属性：

（1）进程既是一个拥有资源的独立单位，它可独立分配虚地址空间、主存和其它系统资源；

（2）进程又是一个可独立调度和分派的基本单位。

这二个基本属性使进程成为并发执行的基本单位。在一些OS中，象大多数UNIX系统、Linux等，进程同时具有这二个属性。而另一些OS中，象WinNT、Solaris、OS／2、Mac OS等，这二个属性由OS独立处理。为了区分二个属性，资源拥有单元称为进程（或任务），调度的单位称为线程。

2.6 线程及其管理

【线程的定义】：

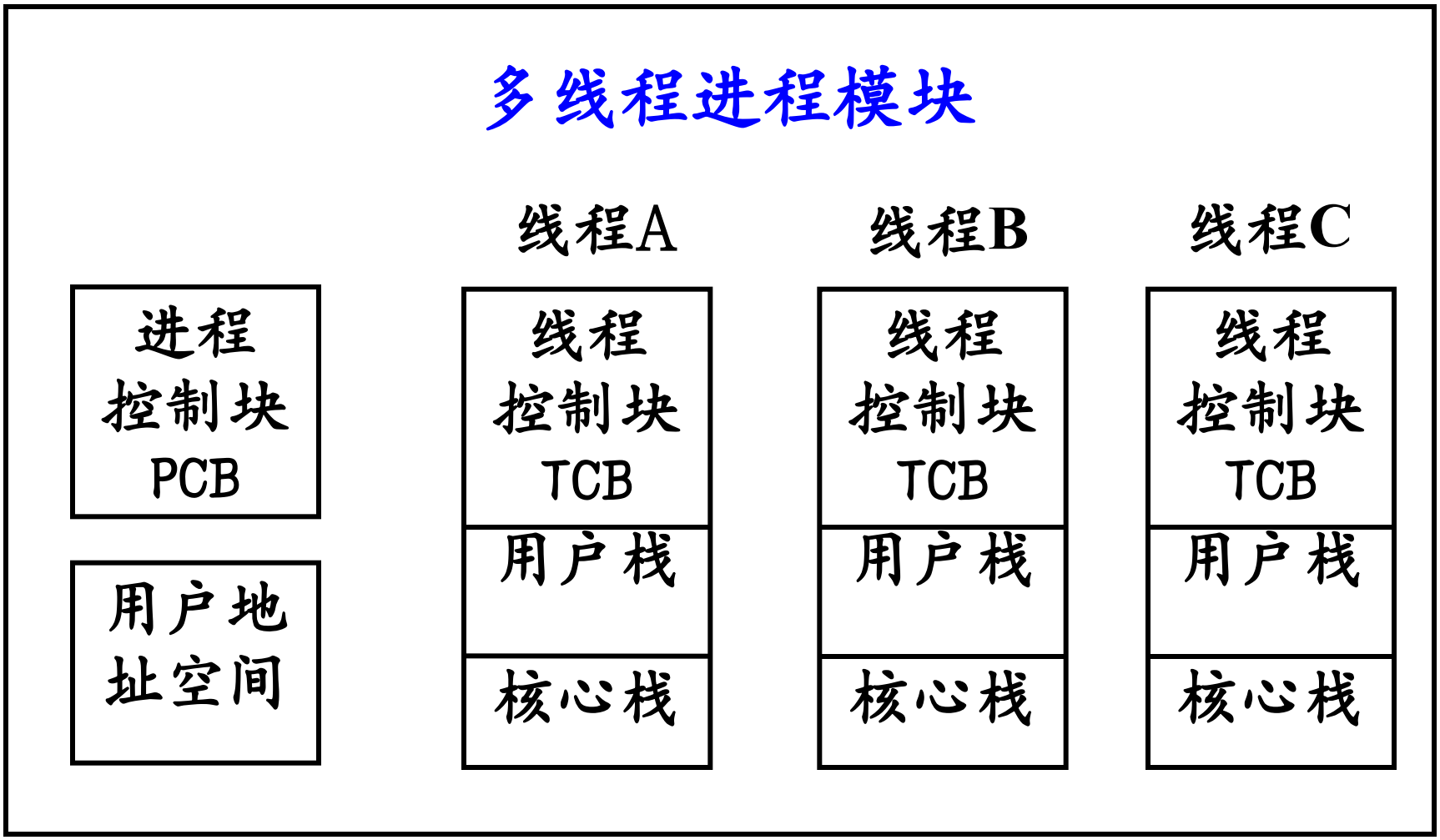
进程内一个执行单元或一个可调度实体。

线程只拥有一点在运行中必不可省的资源（程序计数器、一组寄存器和栈），但它可与同属一个进程的其它线程共享进程拥有的全部资源。

在具有多线程的操作系统中，处理机调度的基本单位是线程。一个进程可以有多个线程，而且至少有一个可执行线程。

多线程模型

多线程是OS在一个进程内支持多个线程的能力。



2.6 线程及其管理

【线程的特征】：

- 创建线程比创建进程快，且节省开销。
- 一个进程至少要有一个可执行线程,可以有多个线程。
- 参与竞争处理机的基本调度单位是线程。
- 线程调度程序是内核的主要成分,也是其主要功能之一。
- 一个线程可以创建它所需的线程。
- 一个线程可以有就绪,等待,运行等状态。
- 方便而有效地实现并行性：进程可创建多个线程来执行同一个程序的不同部分。
- 用线程实现并行性比用进程实现更方便更有效。

2.5 线程及其管理

【进程和线程关系】：

- 线程是进程的一个组成部分。每个进程创建时通常只有一个线程，需要时可创建其他线程。
- 进程的多线程都在进程的地址空间活动。
- 资源是分给进程的，不是分给线程的。线程在执行中需要资源时，可从进程资源中划分。
- 处理机调度的基本单位是线程，线程之间竞争处理机。真正在CPU上运行的是线程。
- 线程在执行时，需要同步。