# 2.4 进程同步

## 课堂练习题：

- 1、有两个用户进程A和B，在运行过程中都要使用系统中的一台打印机输出计算结果。
- 试说明A、B两进程之间存在什么样的制约关系?
- 为保证这两个进程能正确地打印出各自的结果，请用信号量和P、V操作写出各自的有关申请、使用打印机的代码。要求给出信号量的含义和初值。

# 2.4 进程同步

**解：（1）A、B两进程之间存在互斥的制约关系。因为打印机属于临界资源，必须一个进程使用完之后另一个进程才能使用。**

**（2）mutex：用于互斥的信号量，初值为1。**

| 进程A | 进程B |
| --- | --- |
| … | … |
| … | … |
| P(mutex) | P(mutex) |
| 申请打印机 | 申请打印机 |
| 使用打印机 | 使用打印机 |
| V(mutex) | V(mutex) |
| … | … |

# 2.4.5 管程机制

## 1、管程的定义

### (1)管程的引入

访问临界资源的进程自行使用P、V操作会出现误操作，比如：

P、V操作没有成对出现；
只有P，而没有V，或者只有V，而没有P；
P、V操作颠倒；
两个P操作颠倒等。

对此，有人提出能否将对临界资源的控制与管理集中起来，用户只要提出使用要求，其它诸如互斥、同步等问题则不需要用户自己解决，而由系统统一解决。

# 2.4.5 管程机制

1、管程的定义

(1) 管程的引入

(2) 管程的定义

系统中的各种硬件资源和软件资源均可用数据结构抽象地描述其资源特性，即用少量信息和对该资源所执行的操作来表征该资源，而忽略了它们的内部结构和实现细节。

20世纪70年代，Hoare 和B. Hansen提出了一种同步原语，称为管程。当进程共享某临界资源时，只需调用相应的管程，而对资源的使用及其出现的诸如同步、互斥等问题，则完全由管程去解决。

B. Hansen 给管程下的定义是：一个管程指定义了一个数据结构和能为并发程序在该数据结构上执行的一组操作，这组操作能同步进程和改变管程中的数据。

# 2.4.5 管程机制

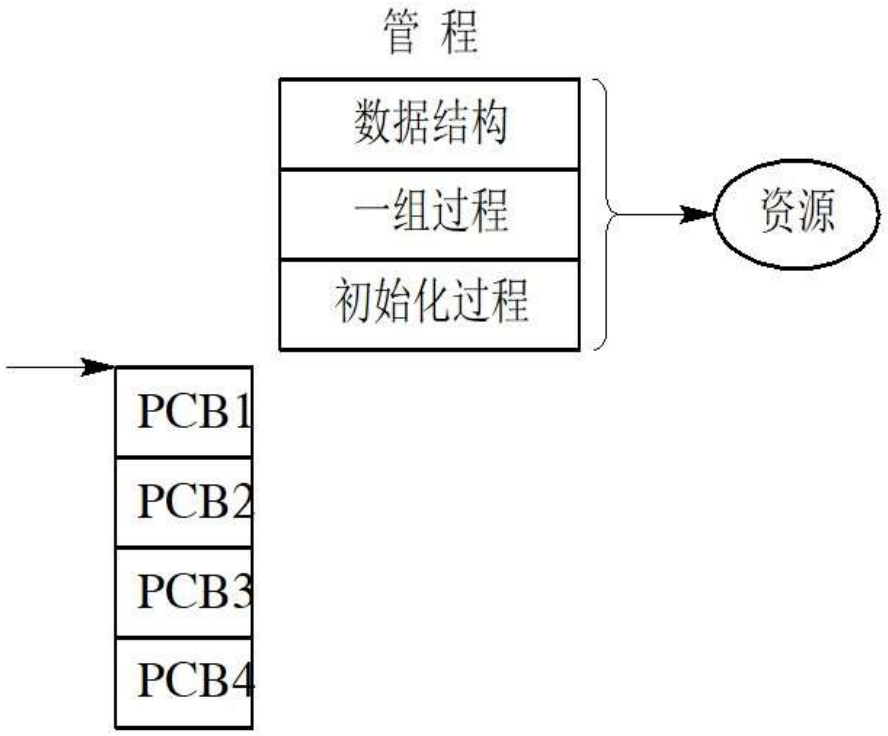
## (3) 管程的组成

(1) 管程标识符；

(2) 局部于管程内部的共享数据结构说明；

(3) 对该数据结构进行操作的一组过程；

(4) 对局部于管程的共享数据设置初始值的语句。




管程的结构

这里用数据结构对资源进行描述，一组过程体现对资源的使用方式。进程在管程外排队等候使用资源。

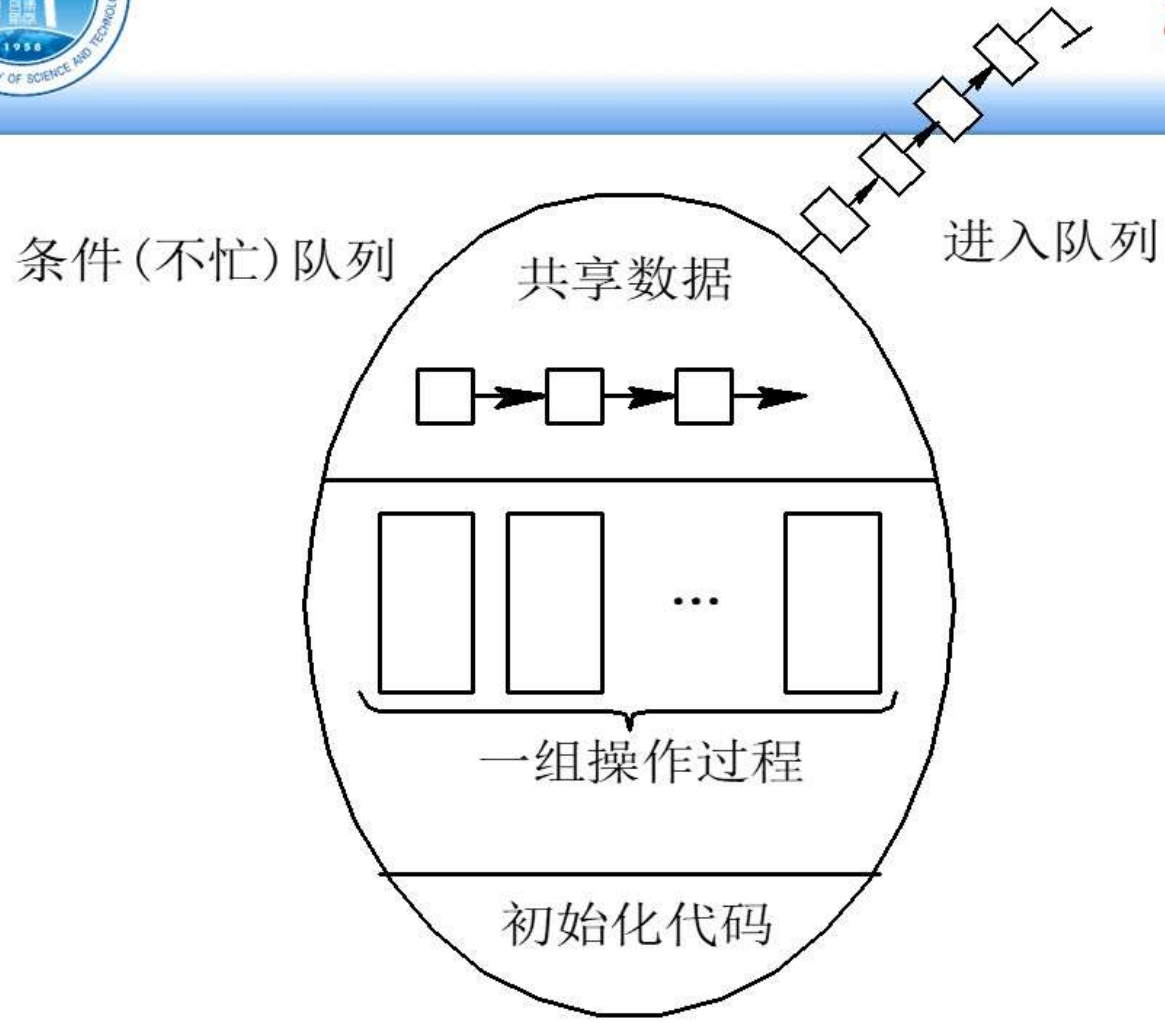



图 2-15 管程的示意图

## (4) 管程的语法

```
MONITOR monitor-name
{
 share variable declarations;  /*<共享变量说明>*/
 cond  declarations; /*<条件变量说明>*/
  public:
  void p1(……)
  {……}
   void p2(……)
   {……}
   ……
    void (……)
    {……}
   { <管程的局部数据初始化语句序列>; }
}
```

# 2.4.5 管程机制

## (5) 管程的特性

管程是一种程序设计语言结构成分，它和信号量有同等的表达能力，从语言的角度看，管程有以下特性：

(1)模块化：基本程序单位，可单独编译；

(2)抽象数据类型：有数据以及对数据的操作；

(3)信息掩蔽：管程内的数据结构以及过程实现外部不可见。

# 2.4.5 管程机制

## (6)管程与进程的差别

(1)进程私有数据结构（PCB），管程公共的；

(2)进程是程序执行，管程是用于同步的机制；

(3)进程目的是并发执行，管程解决公共资源互斥使用；

(4)进程调用管程中的过程，进程主动，管程被动；

(5)进程可以并发，管程不能与调用者并发；

(6)进程具有动态性，有生命期，管程属于OS的资源管理模块，供进程调用的。

# 2.4.5 管程机制

## 1、管程的定义

## 2、条件变量

管程中对每个条件变量，都须予以说明，其形式为：condition x，y；该变量应置于wait和signal之前，即可表示为x.wait和x.signal。

例如，由于共享数据被占用而使调用进程等待，该条件变量的形式为：condition nonbusy;此时， wait原语应改为nonbusy.wait，相应地，signal应改为nonbusy.signal。

应当指出，x.signal操作的作用，是重新启动一个被阻塞的进程，但如果没有被阻塞的进程，则x.signal操作不产生任何后果。这与信号量机制中的signal操作不同。因为，后者总是要执行s:=s+1操作，因而总会改变信号量的状态。

## 2、条件变量

如果有进程Q处于阻塞状态， 当进程P执行了x.signal操作后，怎样决定由哪个进行执行，哪个等待，可采用下述两种方式之一进行处理：

(1) P等待，直至Q离开管程或等待另一条件。
(2) Q等待，直至P离开管程或等待另一条件。

采用哪种处理方式， 当然是各执一词。 但是Hansan却采用了第一种处理方式。Hoare采用了二者折中 ， P执行的x.signal操作是管程中最后的操作。

# 第二章 进程的描述与控制

今日学习任务整理单：

1.理解进程同步、临界资源；
2.掌握记录型信号量；
3. 学会运用信号量解决进程同步问题；
4. 明确进程与管程的区别

进程同步

今日学习资料：

1. 雨课堂中的MOOC资源-第二章进程管理-（5）进程同步；
4. 教材2.4和2.5。

今日作业：理解什么是进程同步。

09:59

105

## 2.5.1 生产者–消费者问题

前面我们已经对生产者—消费者问题(The proceducer-consumer problem)做了一些描述，但未考虑进程的互斥与同步问题，因而造成了数据counter的不定性。由于生产者—消费者问题是相互合作的进程关系的一种抽象，例如， 在输入时，输入进程是生产者，计算进程是消费者；而在输出时，则计算进程是生产者，而打印进程是消费者， 因此，该问题有很大的代表性及实用价值。

## 1. 利用记录型信号量解决生产者—消费者问题

假定在生产者和消费者之间的公用缓冲池中，具有n个缓冲区，这时可利用互斥信号量mutex实现诸进程对缓冲池的互斥使用；利用信号量empty和full分别表示缓冲池中空缓冲区和满缓冲区的数量。只要缓冲池未满，生产者便可将消息送入缓冲池；只要缓冲池未空，消费者便可从缓冲池中取走一个消息。

对生产者—消费者问题可描述如下：

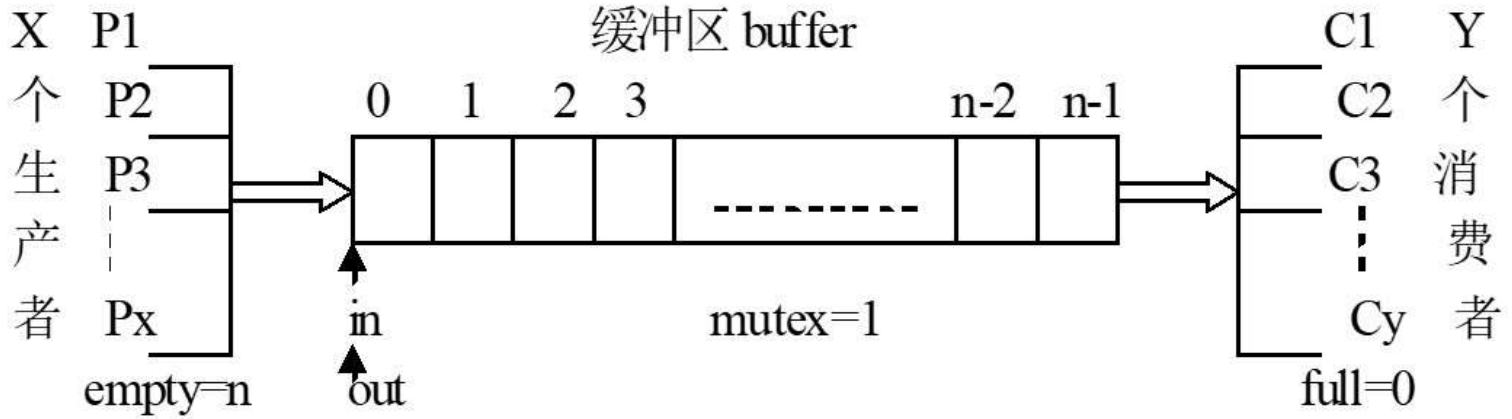



图 2—11 生产者与消费者问题图示

```
int in=out=0;
item buffer[n];
semaphore  mutex=1, empty=n, full=0;
void proceducer()
{
   do{
        ……;
        producer an item nextp;
        wait(empty);
        wait(mutex);
        buffer[in]= nextp;
        in= (in+1) % n;
        signal(mutex);
        signal(full);
       }while(TRUE);
 }
```

```
void consumer()
{
   do{
        wait(full);
        wait(mutex);
        nextc = buffer[out];
        out= (out+1) % n;
        signal(mutex);
        signal(empty);
        consumer the item in nextc;
        ……;
       }while(TRUE)
 }
 void main()
{
  cobegin
  proceducer() ,consumer();
  coend
}
```

# 2.5 经典进程的同步问题

在生产者–消费者问题中应注意：首先，在每个程序中用于实现互斥的**wait(mutex)**和**signal(mutex)**必须成对地出现； 其次，对资源信号量**empty**和**full**的**wait**和**signal**操作，同样需要成对地出现，但它们分别处于不同的程序中。例如，**wait(empty)**在计算进程中，而**signal(empty)**则在打印进程中，计算进程若因执行**wait(empty)**而阻塞，则以后将由打印进程将它唤醒；

最后，在每个程序中的多个**wait**操作顺序不能颠倒。应先执行对资源信号量的**wait**操作，然后再执行对互斥信号量的**wait**操作，否则可能引起进程死锁。

# 2. 利用AND信号量解决生产者—消费者问题

```
int in=out=0;
item buffer[n];
semaphore  mutex=1, empty=n, full=0;
void proceducer()
{
   do{
        ……;
        producer an item nextp;
        Swait(empty, mutex);
        buffer[in]= nextp;
        in= (in+1) % n;
        Ssignal(full, mutex);
       }while(TRUE);
 }
```

```
void consumer()
{
   do{
        Swait(full, mutex);
        nextc = buffer[out];
        out= (out+1) % n;
        Ssignal(mutex, empty);
        consumer the item in nextc;
         ……;
       }while(TRUE)
 }
 void main()
{
  cobegin
  proceducer() ,consumer();
  coend
}
```

## 3. 利用管程解决生产者-消费者问题

在利用管程方法来解决生产者-消费者问题时，首先便是为它们建立一个管程，并命名为Proclucer-Consumer，或简称为PC。其中包括两个过程：

(1) put(item)过程。生产者利用该过程将自己生产的产品投放到缓冲池中，并用整型变量count来表示在缓冲池中已有的产品数目，当count≥n时，表示缓冲池已满，生产者须等待。

(2) get(item)过程。消费者利用该过程从缓冲池中取出一个产品，当count≤0时，表示缓冲池中已无可取用的产品，消费者应等待。

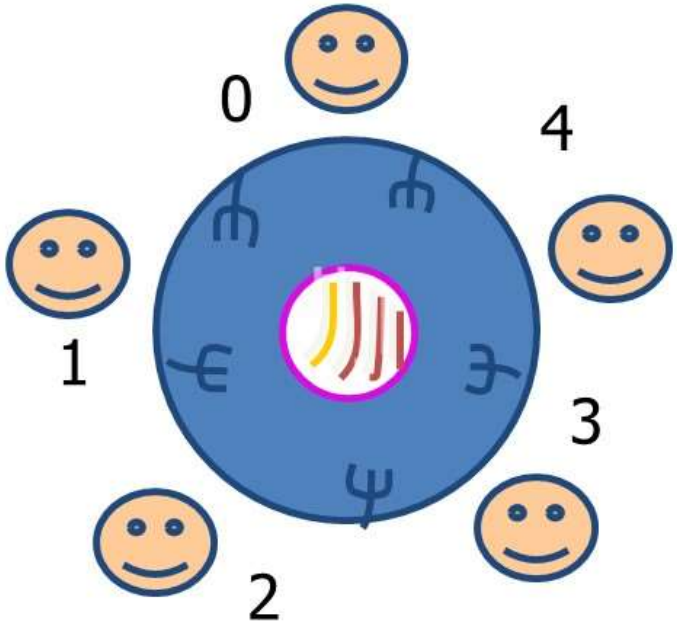
# 2.5 经典进程的同步问题

```
Monitor  producer-consumer
{
    item buffer[N];
    int in,out,count;
    condition  notfull, notempty;
    public:
    void put(item x)
    {
      if (count>=N)  cwait(notfull);
      buffer[in] = x;
      in= (in+1) % N;
      count++;
      csignal(notempty);
      }
```

```
 void get(item *y)
 {
   if (count<=0) cwait(notempty);
   *y= buffer[out];
   out= (out+1) % N;
   count--;
   csignal(notfull);
 }
 {
   in = out = 0; count = 0;
 }
}PC
```

## 1. 利用记录型信号量解决哲学家进餐问题

经分析可知，放在桌子上的筷子是临界资源，在一段时间内只允许一位哲学家使用。为了实现对筷子的互斥使用，可以用一个信号量表示一只筷子，由这五个信号量构成信号量数组。其描述如下：



# 2.5 经典进程的同步问题

```
semaphore  chopstick[5]={1,1,1,1,1};
```

第i位哲学家的活动可描述为：

```
do{
        wait(chopstick[i]);
        wait(chopstick[(i+1)%5]);
        ……
        //eat;
        ……
        signal(chopstick[i]);
        signal(chopstick[(i+1)%5]);
        ……
        //think;
 }while(TRUE);
```

# 2.5 经典进程的同步问题

可采取以下几种解决方法：

(1) 至多只允许有四位哲学家同时去拿左边的筷子，最终能保证至少有一位哲学家能够进餐，并在用毕时能释放出他用过的两只筷子，从而使更多的哲学家能够进餐。

(2) 仅当哲学家的左、右两只筷子均可用时，才允许他拿起筷子进餐。

(3) 规定奇数号哲学家先拿他左边的筷子，然后再去拿右边的筷子；而偶数号哲学家则相反。按此规定，将是1、2号哲学家竞争1号筷子；3、4号哲学家竞争3号筷子。即五位哲学家都先竞争奇数号筷子，获得后，再去竞争偶数号筷子，最后总会有一位哲学家能获得两只筷子而进餐。

## 2. 利用AND信号量机制解决哲学家进餐问题

在哲学家进餐问题中，要求每个哲学家先获得两个临界资源（筷子）后方能进餐，这在本质上就是前面所介绍的AND同步问题，故用AND信号量机制可获得最简洁的解法。

```
semaphore chopstick[5]={1,1,1,1,1};
do{
      ……
      //think;
      ……
      Sswait(chopstick[(i+1)%5], chopstick[i]);
      ……
      //eat;
      …….
      Ssignat(chopstick[(i+1)%5], chopstick[i]);
}while(TRUE);
```

# 2.5.3 读者-写者问题

## 1. 利用记录型信号量解决读者-写者问题


读者与写者问题：

(1)读者可以同时读；(2)读者与写者不能同时；(3)写者之间也不能同时。

为实现Reader与Writer进程间在读或写时的互斥而设置了一个互斥信号量Wmutex。另外，再设置一个整型变量Readcount表示正在读的进程数目。仅当Readcount=0，Reader进程才需要执行Wait(Wmutex)操作。若wait(Wmutex)操作成功，Reader进程便可去读，相应地，做Readcount+1操作。同理，仅当Reader进程在执行了Readcount减1操作后其值为0时，才须执行signal(Wmutex)操作，以便让Writer进程写。又因为Readcount是一个可被多个Reader进程访问的临界资源，因此，应该为它设置一个互斥信号量rmutex。

# 2.5 经典进程的同步问题

读者-写者问题可描述如下：

```
semaphore rmutex=wmutex=1;
int readcount= 0;
void reader()
{
  do{
       wait(rmutex);
       if(readcount==0) wait(wmutex);
       readcount++;
       signal(rmutex);
       ……
       perform read operation;
        ……
       wait(rmutex);
       readcount--;
       if(readcount==0) signal(wmutex);
       signal(rmutex);
   }while(TRUE);
}
```

```
void writer()
{
   do{
        wait(wmutex);
        perform write operation;
        signal(wmutex);
    }while(TRUE);
}

void main()
{
  cobegin
  reader() ;writer();
  coend
}
```

## 2. 利用信号量集机制解决读者-写者问题

```
int RN;
semaphore  L= RN, mx=1;
void reader()
{
  do{
       Swait(L,1,1);
       Swait(mx,1,0);
       …….
       perform read operation;
       ……
       Ssignal(L,1);
  }while(TRUE);
}
```

```
void writer()
{
  do{
       Swait(mx,1,1; L,RN,0);
        perform write operation;
        Ssignal(mx,1);
   }while(TRUE);
}
void main()
{
  cobegin
     reader(); writer();
  coend
}
```

# 主观题 10分

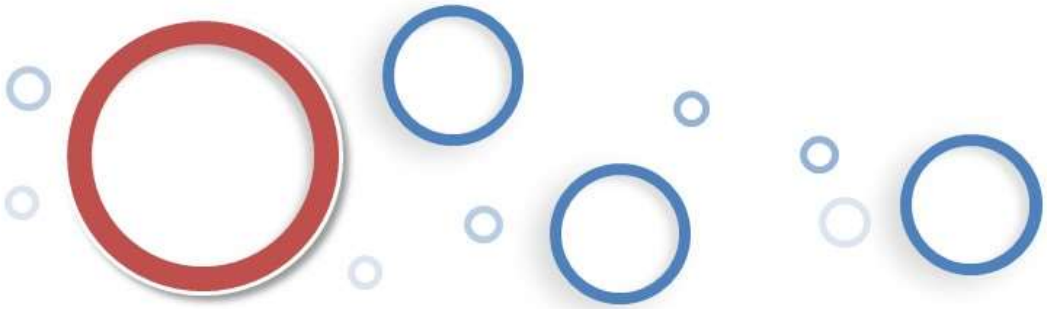
题目：桌上有一空盘，允许存放一只水果。爸爸可向盘中放苹果，也可向盘中放桔子，儿子专等吃盘中的桔子，女儿专等吃盘中的苹果。规定当盘空时一次只能放一只水果供吃者取用，请用P、V原语实现爸爸、儿子、女儿三个并发进程的同步。

11:38

121

# 2.4 进程同步

题目：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。



# 2.4 进程同步

**题目：有一个仓库，可以存放A和B两种产品，但要求：**
**(1)每次只能存入一种产品(A或B)；**
**(2)-N < A产品数量-B产品数量 < M。**
**其中，N和M是正整数。试用P、V操作描述产品A与B的入库过程。**