习题1

在一个盒子里，混装了数量相等的黑白围棋子。现在用自动分拣系统把黑子、白子分开，设分拣系统有两个进程P1和P2，其中P1拣白子，P2拣黑子。规定当一个进程拣了一子后，必须让另一个进程去拣。用信号量和PV操作协调两进程的活动。

灵魂3问：

- 互斥问题or同步问题or资源管控问题？　同步
- 需要几个信号量？　2个，需要循环执行
- 初值如何设置？　分别设置为0和1

解答1

```
struct semaphore S1, S2;
S1.value=1; S2.value=0;
cobegin
    process P1(){
        while(true){
            P(S1);
            拣黑子();
            V(S2);
        }
    }
coend
```

P(S1)即P(S1);

V(S1)即V(S1)

```
process P2(){
    while(true){
        P(S2);
        拣黑子();
        V(S1);
    }
}
coend
```

习题2

某控制系统中，数据采集进程负责把采集到的数据放到一缓冲区中，分析进程负责把数据从缓冲区中取出进行分析，试用信号量实现两者之间的同步。

灵魂3问：

- 互斥问题or同步问题or资源管控问题？ 同步
- 需要几个信号量？ 2个，需要循环执行
- 初值如何设置？ 分别设置为0和1

解答2

```
Struct semaphore S1, S2;
S1.value=1; S2.value=0;
cobegin
    process P1(){
        while(true){
            采集数据();
            P(S1);
            放缓冲区();
            V(S2);
        }
    }
coend
```

```
process P2(){
    while(true){
        P(S2);
        缓冲区中取数据();
        V(S1);
        分析();
    }
}
```

习题3

图书馆规定，每位进入图书馆的读者要在登记表上登记，退出时要在登记表上注销。

（1）用信号量实现读者之间的互斥登记和注销；

（2）图书馆共有100个座位，当图书馆中没有空座位时，后到的读者在图书馆要等待（阻塞）。

灵魂3问：

互斥和资源管控

- 互斥问题or同步问题or资源管控问题？
- 需要几个信号量？

2个，1个互斥，1个资源管控

- 初值如何设置？

分别设置为1和100

解答3

```
Struct semaphore S, mutex;
s.value=100;            /*100个座位资源*/
mutex.value=1;          /*互斥信号量*/
reader(){
    P(s);
    P(mutex);
    进入登记;
    V(mutex);
    读书;
    P(mutex);
    退出登记;
    V(mutex);
    V(s);
}
```

习题4

一家四人父、母、儿子、女儿围桌而坐；桌上有一个水果盘； 当水果盘空时，父亲可以放香蕉或者母亲可以放苹果，但盘中已有水果时，就不能放，父母等待。当盘中有香蕉时，女儿可吃香蕉，否则，女儿等待；当盘中有苹果时，儿子可吃，否则，儿子等待。

解答4

- 果盘为空父母可以操作，果盘非空父母不能操作——需要1个互斥信号量
- 父母操作后，若果盘有香蕉，女儿可以吃——需要1个同步信号量
- 父母操作后，若果盘有苹果，儿子可以吃——需要1个同步信号量

【小结】

共需要1个互斥信号量和2个同步信号量

互斥信号量初值为1；

同步信号量初值为0。

```
Struct semaphore s1, s2, mutex;
s1.value = s2.value = 0;        /*同步信号量*/
mutex.value=1;                  /*互斥信号量*/
```

```
father()
{
    P(mutex);
    放香蕉;
    V(s1);
}
```

```
daughter()
{
    P(s1);
    吃香蕉;
    V(mutex);
}
```

```
son()
{
    P(s2);
    吃苹果;
    V(mutex);
}
```

```
mom()
{
    P(mutex);
    放苹果;
    V(s2);
}
```

习题5

在公共汽车上，司机和售票员的活动分别是：

司机的活动：　　启动车辆
　　正常运行
　　到站停车

售票员的活动：　　关车门
　　售票
　　开车门

在汽车不断的到站、停车和行驶过程中，司机和售票员的活动有什么同步关系？用信号量和P，V操作实现。

- 售票员先关车门后，司机才能启动车辆、正常行驶和到站停车——需要1个同步信号量；
- 售货员关闭车门后就可以开始售票——不需要信号量；
- 司机到站停车后，售票员才能打开车门——需要1个同步信号量；

【小结】共需要2个同步信号量。初值如何设置？

struct semaphore s1, s2;
s1.value = 1;　s2.value = 0;

司机：	售票员：
P(s1);	关车门;
启动车辆;	**V(s1);**
正常运行;	售票;
到站停车;	**P(s2);**
V(s2);	开车门;

习题6

有一个超市，最多可容纳N个人进入购物，当N个顾客满员时，后到的顾客在超市外等待；超市中只有一个收银员。可以把顾客和收银员看作两类进程，两类进程间存在同步关系。写出用P、V操作实现的两类进程的算法。

解法1

超市能容纳N个顾客——需要1个资源信号量s；

- 顾客购物后可以找收银员结账——需要1个同步信号量s1；
- 收银员收银结束后顾客可以离开——需要1个同步信号量s2；

【小结】共需要3个信号量，1个资源s，2个同步信号量s1和s2。

```
struct semaphore s, s1, s2;
s.value = N;            /*s用于资源*/
s1.value = 0;           /*s1用于顾客结账前和收银员的同步*/
s2.value = 0;           /*s2用于收银员结账后和顾客的同步*/
```

解法1

源信号量s;

——需要1个同步信号

顾客：

```
P(s);
进入超市;
购物;
V(s1);      /*通知收银员收银*/
结账;
P(s2);      /*判断是否收银结束*/
V(s);   /*离开超市，下一位进入*/
```

收银员：

```
P(s1);      /*判断是否可以收银*/
收银;
V(s2);      /*收银结束，顾客可以离开*/
```

```
struct semaphore s, s1, s2;
s.value = N;            /*s用于资源*/
s1.value = 0;           /*s1用于顾客结账前和收银员的同步*/
s2.value = 0;           /*s2用于收银员结账后和顾客的同步*/
```

解法2

- 超市能容纳N个顾客——需要1个资源信号量s；
- 顾客购物后可以找收银员结账——需要1个同步信号量s1；
- 收银员同一时间只能为1个顾客结账——需要1个互斥信号量mutex；
- 收银员结账过后顾客才能离开——需要1个同步信号量s2；

【小结】共需要4个信号量，1个资源s、1个互斥mutex和2个同步信号量s1和s2。

```
struct semaphore s, s1,s2, mutex;
s.value = N;          /*s用于资源*/
s1.value = 0;         /*s1用于顾客和收银员的同步*/
s2.value = 0;         /*s2用于收银员和顾客的同步*/
mutex = 1;            /*mutex用于收银员互斥*/
```

解法2

资源信号量s；

需要1个同步信号

```
顾客：
    P(s);
    进入超市;
    购物;
    P(mutex);  /*争夺结账权限*/
    V(s1);     /*通知收银员收银*/
    结账;
    P(s2);     /*已结过账，可以离开*/
    V(s);
```

```
收银员：
    P(s1);
    收银;
    V(mutex);  /*通知下一个顾客结账*/
    V(s2);     /*已结账的顾客可以离开*/
```

```
struct semaphore s, s1,s2, mutex;
s.value = N;          /*s用于资源*/
s1.value = 0;         /*s1用于顾客和收银员的同步*/
s2.value = 0;         /*s2用于收银员和顾客的同步*/
mutex = 1;            /*mutex用于收银员互斥*/
```

WARNING

为了便于阅读和编写程序，从本节课开始将wait(S)替换为P(S)，将signal(S)替换为V(S)。

2.5　经典进程的同步问题

在多道程序环境下，进程同步问题十分重要，引起了不少学者对它进行研究，由此产生了一系列经典的进程同步问题，其中较有代表性的是：

生产者-消费者问题

哲学家进餐问题

读者-写者问题

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概念及实现方法。

2.5.1 生产者-消费者问题

前面已经对生产者-消费者问题做了一些描述，但是未考虑进程的互斥和同步问题，因而造成了共享变量counter的不确定性。
生产者-消费者问题是相互合作进程关系的一种抽象，例如，……

生产者-消费者问题从特殊到一般(从易到难)可以分3种形式：

- ▲一个生产者、一个消费者、一个缓冲区的问题；
- ▲一个生产者、一个消费者、n个缓冲区的问题；
- ▲ k个生产者、m个消费者、n个缓冲区的问题；

先介绍最简单的P-C问题

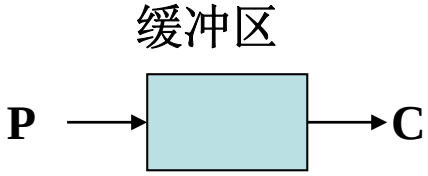
未考虑同步和互斥的生产者-消费者问题

```
producer( ){
while(1){
produce an item in nextp；
…
while (counter==n) ;
buffer[in]=nextp;
in=(in+1) % n;
counter=counter+1;
}}
```

```
consumer( ){
while(1){
while (counter==0 );
nextc=buffer[out];
out=(out+1) % n;
counter=counter-1;
consumer the item in nextc;
}}
```

最简单的生产者-消费者问题

一个生产者、一个消费者、一个缓冲区的问题如右图所示。



★当缓冲区空时，生产者可将产品存入缓冲区；当缓冲区满时，生产者必须等待 (阻塞)，待消费者取走产品后将其唤醒后，才能将产品存入。

★当缓冲区满时，消费者可从缓冲区取出产品进行消费；当缓冲区空时，消费者必须等待(阻塞)，待生产者存入产品后将其唤醒后，才能再从缓冲区取产品。

用信号量机制解决进程同步问题的基本方法：

1．设置2个信号量：1个信号量empty表示空缓冲区的数量，其初值为1，表示有1个空缓冲区；1个信号量full表示满缓冲区的数量，其初值为0，表示开始时没有满缓冲区；（由物理意义确定）

2．生产者将产品存入缓冲区之前，应先测试缓冲区是否空：执行P(empty)操作；离开临界区(存入产品)后，应通知(可能会唤醒)消费者：执行V(full)操作；

3．消费者从缓冲区取产品之前，应先测试缓冲区是否满：执行P(full)操作；离开临界区(取走产品)后，应通知(可能会唤醒)生产者：执行V(empty)操作

循环同步

一个生产者、一个消费者、一个缓冲区的生产者-消费者问题的算法描述如下所示：

```
struct semaphore empty,full;
empty.value=1;full.value=0;

process Producer()
{ ...
   produce an item in nextp;
   P(empty);//测试是否满
   buffer=nextp;
   V(full);//通知消费者可以消费
}
```

```
process Consumer()
{
   P(full); //测试是否空
   nextc=buffer;
   V(empty); //通知生产者可以生产
   consume the item  in nextc;
}
```

一个生产者、一个消费者、n个缓冲区的P-C问题

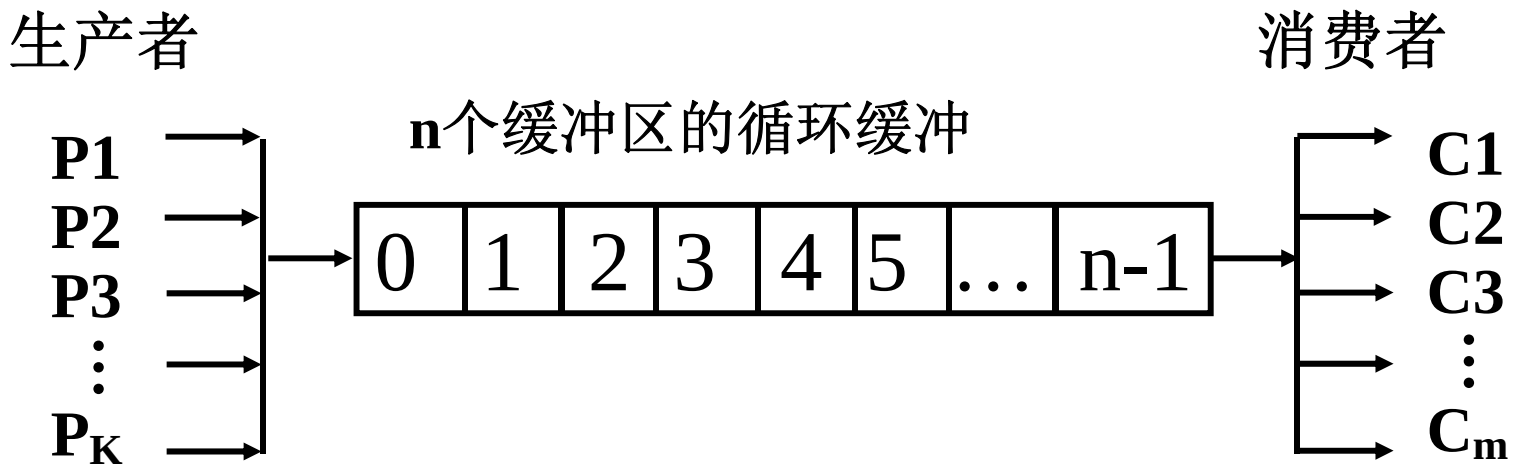

```
struct  semaphore empty, full, mutex;
empty.value=n;full.value=0;
mutex.value=1;
int in=0,out=0;   //下标
process Producer()
{ ...
   produce an item in nextp;
   P(empty);//测试
P(mutex);
   buffer[in]=nextp;
   in=(in+1)%n;
V(mutex);
   V(full);//满的缓冲区加1  }
```

与前不同

```
process Consumer()
{
   P(full); //测试
P(mutex);
   nextc=buffer[out];
   out=(out+1)%n;
V(mutex);
   V(empty); //空的缓冲区加1
   consume the item in nextc;
}
```

下面介绍生产者-消费者问题一般形式：k个生产者、m个消费者、n个缓冲区的问题。

一般形式的生产者-消费者问题的图示如下：



◆用互斥信号量mutex对缓冲区(共享变量in和out)的互斥使用，互斥信号量mutex初值为1；

◆用资源信号量empty表示多缓冲中空缓冲区的数目，empty的初值为n；

◆用资源信号量full表示多缓冲中满缓冲区的数目，full的初值为0；

◆只要多缓冲未满，生产者便可将消息送入缓冲区；

◆只要多缓冲不空，消费者便可从缓冲区取走一个消息。

◆生产者用共享变量in作为下标访问缓冲区，mutex为其互斥信号量；消费者用共享变量out作为下标访问缓冲区，其互斥信号量也用mutex。

生产者-消费者问题可描述如下：

```
semaphore mutex,empty,full ;
item buffer[n] ;
int in = 0， out = 0 ;
mutex.value = 1;
empty.value = n， full.value = 0;
```

初始化

```
parbegin //并发执行开始
process producer_i (i=1,2,…,k）
{
    item nextp ;
    while （condition）
   {  …
      produce an item in nextp;
      P（empty）;
      P（mutex）;
      buffer[in] = nextp ;
      in = （in + 1）% n ;
      V（mutex）;
      V（full）;
   }
}
```

临界区

```
process consumer_j (j=1,2,…,m）
{  item nextc ;
    while （condition）
   {  P（full）;
      P（mutex）;
      nextc = buffer[out] ;
      out = （out + 1）% n ;
      V（mutex）;
      V（empty）;
      consume the item in nextc ;
   }
}
parend //并发执行结束
```

临界区

- 在每个进程中，实现互斥的P（mutex）和V（mutex）必须成对出现；
- 对资源信号量empty和full的P和V操作也要成对地出现，但它们处于不同的进程中。
- 在每个进程中的多个P操作顺序不能颠倒，应先执行对资源信号量的P操作，然后执行对互斥信号量的P操作，否则可能引起进程死锁。

利用AND信号量解决生产者-消费者问题

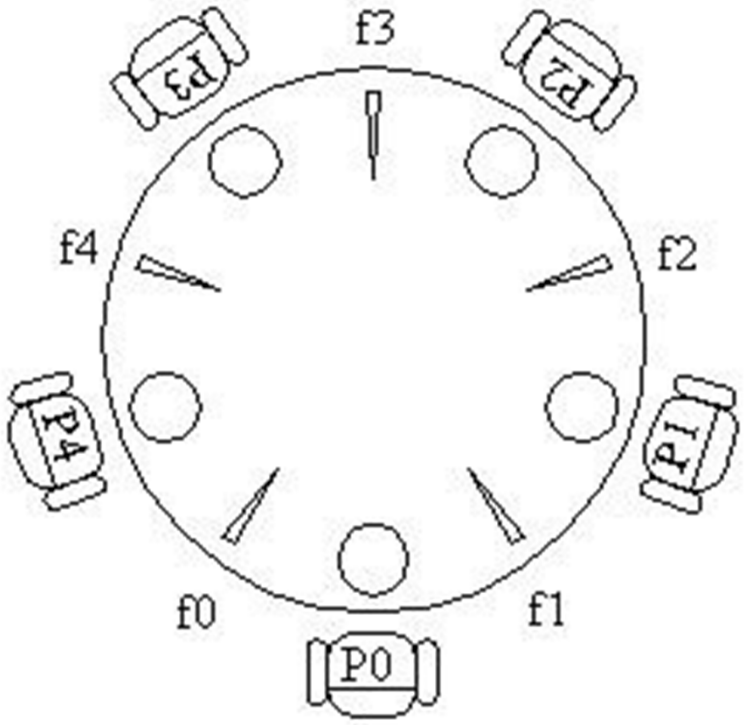
```
semaphore mutex,empty,full ;
item buffer[n] ;
int in = 0，out = 0 ;
mutex.value = 1;
empty.value = n，full.value = 0;
parbegin //并发执行开始
process producer_i (i=1,2,…,k）
{
    item nextp ;
    while （condition）
   {  …
      produce an item in nextp;
      Swait(empty，mutex)
      buffer[in] = nextp ;
      in = （in + 1）% n ;
      Ssignal(empty，mutex)
   }
}
```

```
process consumer_j (j=1,2,…,m）
{  item nextc ;
   while （condition）
   {
      Swait(full，mutex)
      nextc = buffer[out] ;
      out = （out + 1）% n ;
      Ssignal(full，mutex)
      consume the item in nextc ;
   }
}
parend //并发执行结束
```

2.5.2 哲学家进餐问题

哲学家就餐问题描述如下：

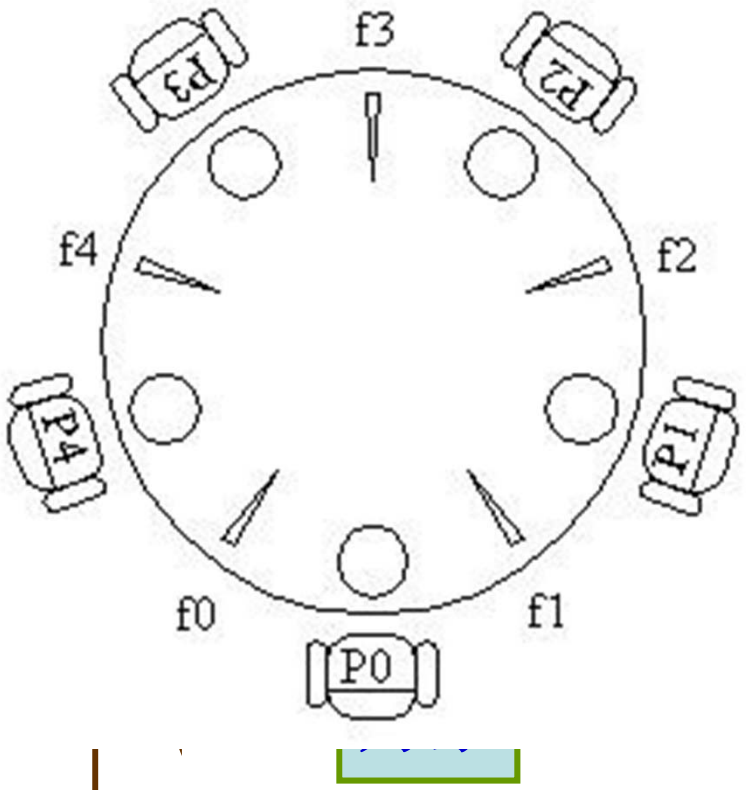
有5个哲学家共用一张圆桌，分别坐在周围的5张椅子上，在圆桌上有5个碗和5只筷子，他们的生活方式是交替地进行思考和进餐。平时，每个哲学家进行思考，饥饿时便试图拿起其左右最靠近他的筷子，只有在他拿到两只筷子时才能进餐。进餐完毕，放下筷子继续思考。



利用记录型信号量解决哲学家进餐问题

桌子上的筷子f0，f1，…，f4是临界资源，应互斥使用，可用一个信号量表示一只筷子，5只筷子的5个信号量构成信号量数组，所有信号量的初值均为1。

```
struct semaphore chopstick[5] ;
chopstick[0].value=chopstick[1].va
chopstick[2].value=chopstick[3].va
chopstick[4].value=1 ;
```



(1) 拿起左、右筷子；
(2) 吃饭；
(3) 放下左、右筷子；
(4) 思考问题；
(5) 返回(1)。

第i个哲学家的活动可描述如下：

```
semaphore  chopstick[5] = {1,  1,  1,  1,  1 }
parbegin
process  Pi（i = 0，  1，  2，  3，  4）
{
   while （true）
   { P（chopstick[i]）;                          //拿起左边筷子
     P（chopstick[（i + 1）% 5]）;               //拿起右边筷子
     eating ;
     V（chopstick[i]）;                          //放下左边筷子
     V（chopstick[（i+1）% 5]）;                 //放下右边筷子
     thinking ;
   }
}
parend
```

此算法虽然能保证相邻哲学家对筷子的访问互斥，但可能引起死锁。(Why?)

对上述哲学家算法的死锁问题，可采取下面几种解决方法之一：

（1）至多允许4个哲学家同时取左边的筷子，这样能至少保证一个哲学家能就餐，并在用毕后释放他用过的两只筷子，从而使更多的哲学家能够进餐。（先自行思考）

（2）仅当哲学家左右两只筷子均可用时，才允许他拿起筷子进餐。 用AND信号量机制

（3）规定奇数号哲学家先拿左边筷子，然后再拿右边筷子；而偶数号哲学家先拿右边筷子，然后再拿左边筷子。

最多允许4个哲学家同时进餐：

算法：

```
semaphore n = 4;
semaphore  chopstick [0…4]  = {1,  1,  1,  1,  1};
Begin
      Parbegin
            Wait(n);
                  Wait(chopstick[i]);
                  Wait(chopstick[(i+1) mod 5]);
                  …
                  Eat;
                  …
                  V(chopstick[i]);
                  V(chopstick[(i+1) mod 5]);
            V(n);
            think;
      parend;
end.
```

奇数号哲学家先拿其左边的筷子，然后再拿其右边的筷子；
偶数号哲学家先拿其右边的筷子，然后再拿其左边的筷子。

```
semaphore  chopstick [0…4]  = {1,  1,  1,  1,  1};
Parbegin
  If (i mod 2 !=0) then
       begin
            Wait(chopstick[i]);
            Wait(chopstick[(i+1) mod 5]);
       End
  Else
       begin
            Wait(chopstick[(i+1) mod 5]);
            Wait(chopstick[i]);
       End
  …
  Eat;
  …
  V(chopstick[i]);
  V(chopstick[(i+1) mod 5]);
  think;
  until false;
parend;
```

2.5.3 读者-写者问题

一个数据文件或记录，可被多个进程共享，我们把只要求读该文件的进程称为“读者进程”，其他进程称为“写者进程”。

- 允许多个读者进程同时读一个共享文件，因为读操作不会使数据文件混乱；
- 不允许两个或两个以上写者进程同时访问共享文件，因为这种访问将会引起混乱；
- 不允许一个写者进程和其他读者进程同时访问共享文件，因为这种访问将会引起混乱。

所谓“读者-写者问题”——是指保证一个**Writer**进程必须与其它进程互斥地访问共享对象的同步问题。

利用记录型信号量解决读者-写者问题

▲ 为实现Reader进程和Writer进程间的互斥，设置一个互斥信号量wmutex，其初值为1；

▲ Writer进程需要执行P(wmutex)操作；

▲ 由于只要有一个Reader进程在读，便不允许Writer进程去写，因此只有第一个读者进程需要执行P(wmutex)操作；

▲ 设置一个整型变量readcount，记录正在读的读者进程数，其初值为0；

▲ 当readcount =0时，Reader进程才需要执行P(wmutex)操作；

▲ 若P(wmutex)操作成功（表示此时无Writer进程在写），Reader进程便可去读，同时做readcount+1的操作。

【算法分析】

▲ 同理，最后一个Reader进程离开时(readcount-1后变为0)应执行V(wmutex)操作，以便让Writer进程写。

▲ readcount是被多个Reader进程访问的临界资源，为了对它互斥访问，应为它设置一个互斥信号量rmutex。

根据前面分析，读者-写者问题可描述如下：

```
semaphore wmutex，rmutex;
int  readcount = 0;
wmutex.value=rmutex.value=1;
parbegin
process Reader_i (i = 1，2，…)
{
   P(rmutex);
   if(readcount==0)
      P(wmutex);
   readcount = readcount + 1;
   V(rmutex);
   …
   Reading;
   …
   P(rmutex);
   readcount = readcount – 1;
   if(readcount==0)
      V(wmutex);
   V(rmutex);
}
```

```
process Writer_j (j=1，2，…)
{
   P(wmutex);
      Writing;
   V(wmutex);
}
parend
```

读者优先

【分析】▲当第一个读者在读文件时，后续读者也可进入临界区读该文件，后续写者不能写(在wmutex上阻塞)；待所有读者退出时，由最后退出的读者唤醒一个写者。

▲当有一个写者在写时，后续写者不能写，在wmutex上阻塞；后续读者不能读，其中第一个读者在wmutex上阻塞，其余读者在rmutex上阻塞。该写者退出时，唤醒一个写者或读者。

习题：

有一座东西方向的独木桥；用P、V操作分别实现以下要求：

（1） 每次只允许一个人过桥；

（2）当独木桥上有自东向西的行人时，同方向的行人可以同时过桥，从西向东的方向，只允许一个人单独过桥。

（3）当独木桥上有行人时，同方向的行人可以同时过桥，相反方向的人必须等待。