第 3 章　应用题参考答案

1. 有三个并发进程：R 负责从输入设备读入信息块，M 负责对信息块加工处理；P 负责打印输出信息块。今提供：(1) 一个缓冲区，可放置 K 个信息块；(2)二个缓冲区，每个可放置 K 个信息块；试用信号量和 PV 操作写出 3 个进程正确工作的流程。

答：

(1)
```
item B[k];
semaphore sread=k;
semaphore smanage=0;
semaphore swrite=0;
int rptr=0;
int mptr=0;
int wptr=0;
item x;
  cobegin
process reader ( ) {
  while(true) {
read a message into x ;
  P(sread);
  B[rptr]=x;
  rptr= (rptr+1) % k;
  V(smanage);
  }
}
process manager( ) {
  while(true) {
  P(smanage);
  x=B[mptr];
mptr=(mptr+1) % k;
manage the message in x ;
  B[mptr]=x;
  V(swrite);
  }
}
process writer ( ) {
  whilr(true) {
  P(swrite);
  x=B[wptr];
 wptr=(wptr+1) % k;
  V(sread);
print the message in x;
  }
}
coend
```

(2)
```
item A[k],B[k];
semaphore sput1=k ;
semaphore sput2=k ;
semaphore sget =0 ;
semaphore sget2=0 ;
int put1=0 ;
int put2=0 ;
int get1=0 ;
int get2=0 ;
 cobegin
process reader( ) {
  while(true) {
read a message into x ;
  P(sput1);
  A[put1]=x;
  put1=(put1+1) % k ;
process manager( ) {
  while(true) {
  P(sget1) ;
  x =A[get1];
  get1=(get1+1) % k;
  V(sput1) ;
process writer( ) {
  while(true) {
  P(sget2);
  x=B[get2];
  get2 =(get2+1) % k;
  V(sput2);
```

```
   V(sget1);                Manage the message into x;      Print the message in x;
   }                        P(sput2);                       }
}                           B[put2]=x;                   }
                            put2=(put2+1) % k ;
                            V(sget2);
                            }
                           }
 coend
```

2. 设有 n 个进程共享一个互斥段，如果：(1)每次只允许一个进程进入互斥段；(2)每次最多允许 m 个进程（m≤n）同时进入互斥段。试问：所采用的信号量初值是否相同？信号量值的变化范围如何？

答：所采用的互斥信号量初值不同。

(1) 互斥信号量初值为 1，变化范围为[-n+1，1]。

当没有进程进入互斥段时，信号量值为 1；当有 1 个进程进入互斥段但没有进程等待进入互斥段时，信号量值为 0；当有 1 个进程进入互斥段且有一个进程等待进入互斥段时，信号量值为-1；最多可能有 n-1 个进程等待进入互斥段，故此时信号量的值应为-(n-1)也就是-n+1。

(2) 互斥信号量初值为 m，变化范围为[-n+m，m]。

当没有进程进入互斥段时，信号量值为 m；当有 1 个进程进入互斥段但没有进程等待进入互斥段时，信号量值为 m-1；当有 m 个进程进入互斥段且没有一个进程等待进入互斥段时，信号量值为 0；当有 m 个进程进入互斥段且有一个进程等待进入互斥段时，信号量值为-1；最多可能有 n-m 个进程等待进入互斥段，故此时信号量的值应为-(n-m)也就是-n+m。

3. 有两个优先级相同的进程 P1 和 P2，各自执行的操作如下，信号量 S1 和 S2 初值均为 0。试问 P1、P2 并发执行后，x、y、z 的值各为多少？

```
P1( ) {                     P2( ) {
y=1;                        x=1;
y=y+3;                      x=x+5;
V(S1);                      P(S1);
z=y+1;                      x=x+y;
P(S2);                      V(S2);
y=z+y;                      z=z+x;
}                           }
```

答：现对进程语句进行编号，以方便描述。

```
y=1;        ①               x=1;        ⑤
y=y+3;      ②               x=x+5;      ⑥
V(S1);                      P(S1);
z=y+1;      ③               x=x+y;      ⑦
P(S2);                      V(S2);
y=z+y;      ④               z=z+x;      ⑧
```

①、②、⑤和⑥是不相交语句，可以任何次序交错执行，而结果是唯一的。接着无论系统如何调度进程并发执行，当执行到语句⑦时，可以得到 x=10，y=4。按 Bernstein 条件，语句③的执行结果不受语句⑦的影响，故语句③执行后得到 z=5。最后，语句④和⑧并发执行，

这时得到了两种结果为：

语句④先执行：x=10，y=9，z=15。

语句⑧先执行：x=10，y=19，z=15。

此外，还有第三种情况，语句③被推迟，直至语句⑧后再执行，于是依次执行以下三个语句：

```
z=z+x;
z=y+1;
y=z+y;
```

这时 z 的值只可能是 y+1=5，故 y=z+y=5+4=9，而 x=10。

第三种情况为：x=10，y=9，z=5。

4．现有五个语句，S1:a=5-x; S2:b=a*x; S3:c=4*x; S4:d=b+c; S5:e=d+3;试用 Bernstein 条件证明语句 S2 和 S3 可并发执行，而语句 S3 和 S4 不可并发执行。

答：由于 R(S2)={a,x},W(S2)={b},R(S3)={x},W(S3)={c},故有

$R(S2)\cap W(S3)\cup R(S3)\cap W(S2)UW(S2)\cap W(S3)=\{\ \}$，语句 S2 和 S3 可并发执行。

由于 R(S3)={x},W(S3)={c},R(S4)={b,c},W(S4)={d},故有

$R(S3)\cap W(S4)\cup R(S4)\cap W(S3)UW(S3)\cap W(S4)=\{c\}$，语句 S3 和 S4 不可并发执行。

5．有一阅览室，读者进入时必须先在一张登记表上登记，该表为每一座位列出一个表目，包括座号、姓名，读者离开时要注销登记信息；假如阅览室共有 100 个座位。试用：(1）信号量和 P、V 操作；(2）管程，来实现用户进程的同步算法。

答：(1) 使用信号量和 PV 操作：

```
struct {char name[10] ;
        int number;
    }A[100];
    semaphore mutex,seatcount;
    int i;mutex=1;seatcount=100;
  for(int i=0;i<100;i++ )
    {A[i].number=i;A[i].name=null;}
cobegin
process   reader i(char readername[ ]) {                    //(i=1,2,…)
        P(seatcount);
        P(mutex);
        for (int i=1; i< 100 ;i++)
            if (A[i].name==null ) A[i].name=readername;
        reader get the seat number =i;                  /*A[i].number*/
        V(mutex)
        /*进入阅览室，座位号 i，座下读书*/;
        P(mutex);
        A[i].name=null;
        V(mutex);
        V(seatcount);
        /*离开阅览室*/;
```

```
            }
    coend.
```

(2) 使用管程实现：

```
type readbook=MONTOR {
    cond R;R=0;
    int R_count,i,seatcount;
    char name[100] ;
    R_count =seatcount=0;
    InterfaceModule IM;
    DEFINE readercome( ),readerleave( );
    USE enter(), leave(),wait( ), signal( );
Procedure    readercome(char readername[ ]) {
        enter (IM);
        if (seatcount>=100)    wait(R,R_count,IM);
        seatcount=seatcount+1;
        for (int i=0;i<100;i++) {
            if (name[i]==null )    name[i]=readername;
        }
        get the seat number=i;
        leave(IM );
  }
procedure    readerleave(char readername) {
      enter(IM);
      seatcount--;
      for(int i=0;i<100;i++)
          if (name[i]==readername)    name[i]=null;
      signal(R,R_count,IM);
      leave(IM);
}
    cobegin
    process reader i ( ) {            //i=1,2.…
          readbook.readercome(readername);
          read the book;
          readbook.readerleave(readername);
          leave the readroom;
}
coend
```

6．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，设分拣系统有二个进程 P1 和 P2，其中 P1 拣白子；P2 拣黑子。规定每个进程每次拣一子；当一个进程在拣时，不允许另一个进程去拣；当一个进程拣了一子时，必须让另一个进程去拣。试写出两进程 P1 和 P2 能并发正确执行的程序。

答 1：实质上是两个进程的同步问题，设信号量 S1 和 S2 分别表示可拣白子和黑子，不失一般性，

若令先拣白子。

```
    semaphore S1,S2;
    S1=1;S2=0;
 cobegin
     process P1( ) {
          while(true) {
           P(S1);
           /*拣白子*/
            V(S2);
          }
        }
     process P2( ) {
          while(true) {
           P(S2);
           /*拣黑子*/
            V(S1);
          }
        }
coend
```

答 2：

```
type pickup_chess= MONITOR    {
    bool flag; flag=true;
    cond S_black,S_white; S_black=S_white=0;
    int S_black_count,S_white_count; S_black_count=S_white_count=0;
    InterfaceModule IM;
   DEFINE pickup_black ,pickup_white
   USE enter,leave,wait,signal；
 procedure pickup_black ( ) {
          enter(IM);
           if (flag) wait(S_black,S_black_count,IM);
           flag=true;
           pickup a black;
           signal(S_white,S_white_count,IM);
          leave(IM );
    }
procedure pickup_white( ) {
          enter(IM);
           if(!flag) wait(S_white,S_white_count,IM);
           flag=false;
           pickup a white;
           signal(S_black,S_black_count,IM);
          leave(IM);
    }
```

```
        cobegin
          process_B( ); process_W( );
        coend
    process_B( ) {
          pickup_chess.pickup_black( );
          other;
       }
  process_W( ) {
          pickup_chess.pickup_white( );
          other;
    }
```

7．管程的同步机制使用条件变量和 Wait 及 Signal，尝试为管程设计一种仅仅使用一个原语操作的同步机制。

答：可以采用形如 waituntil <条件表达式>的同步原语。如 waituntil (numbersum+number<K) 表示进程由于条件不满足而应等待，当进程号累加和小于 K 时，系统应唤醒该进程工作。

8．设公共汽车上，司机和售票员的活动分别如下：

司机的活动：启动车辆；正常行车；到站停车。

售票员的活动：关车门；售票；开车门。

在汽车不断地到站、停车、行驶过程中，这两个活动有什么同步关系？用信号量和 P、V 操作实现它们的同步。

答：在汽车行驶过程中，司机活动与售票员活动之间的同步关系为：售票员关车门后，向司机发开车信号，司机接到开车信号后启动车辆，在汽车正常行驶过程中售票员售票，到站时司机停车，售票员在车停后开门让乘客上下车。因此，司机启动车辆的动作必须与售票员关车门的动作取得同步；售票员开车门的动作也必须与司机停车取得同步。

应设置两个信号量：s1、s2；s1 表示是否允许司机启动汽车(其初值为 0)；s2 表示是否允许售票员开门(其初值为 0)。用 P、V 原语描述如下：

```
    semaphore s1，s2；
    s1=0;s2=0;
    cobegin
      driver (    ); busman (    );
    coend
process driver ( ) {
        while(true) {
          P(s1);
          /*启动车辆*/；
          /*正常行车*/；
          /*到站停车*/；
         V(s2);
        }
```

```
   }
process busman (    ) {
              while(true)    {
               /*关车门*/;
               V(s1);
               /*售票*/;
               P(s2);
              /*开车门*/;
              /*上下乘客*/;
          }
     }
```

9．一个快餐厅有 4 类职员：（1）领班：接受顾客点菜；（2）厨师：准备顾客的饭菜；（3）打包工：将做好的饭菜打包；（4）出纳员：收款并提交食品。每个职员可被看作一个进程，试用一种同步机制写出能让四类职员正确并发运行的程序。

答：典型的进程同步问题，可设四个信号量 S1、S2、S3 和 S4 来协调进程工作。

```
      semaphore S1,S2,S3,S4;
      S1=1;S2=S3=S4=0;
cobegin
      process P1( ) {
            while(true) {
             /*有顾客到来*/;
             P(S1);
             /*接受顾客点菜/*;
             V(S2);
             }
          }
      process P2( ) {
             while(true) {
             P(S2);
             /*准备顾客的饭菜*/;
             V(S3);
             }
          }
      process P3( ) {
            while(true) {
             P(S3);
             /*将做好的饭菜打包*/;
             V(S4);
             }
          }
      process P4( ) {
             while(true) {
             P(S4);
```

```
        /*收款并提交食品*/;
        V(S1);
        }
      }
coend
```

10. (1)两个并发进程并发执行，其中，A、B、C、D、E 是原语，试给出可能的并发执行路径。

```
process P( ) {          process Q( ) {
  A;                      C;
  B;                      D;
  C;                     }
  }
```

(2) 两个并发进程 P1 和 P2 并发执行，它们的程序分别如下：

```
P1( ) {                 P2( ) {
 While(true) {           while(true) {
 k=k*2;                  print k;
 k=k+1;                   k=0;
 }                         }
}                         }
```

若令 k 的初值为 5，让 P1 先执行两个循环，然后，P1 和 P2 又并发执行了一个循环，写出可能的打印值，指出与时间有关的错误。

答：

(1)共有 10 种交错执行的路径：

A、B、C、D、E；A、B、D、E、C；A、B、D、C、E；

A、D、B、E、C；A、D、B、C、E；A、D、E、B、C；

D、A、B、E、C；D、A、B、C、E；D、A、E、B、C；D、E、A、B、C。

(2)把语句编号，以便于描述：

k=k*2; ① print k; ③

k=k+1; ② k=0; ④

1) K 的初值为 5，故 P1 执行两个循环后，K=23。
2) 语句并发执行有以下情况：

①、②、③、④，这时的打印值为：47

③、④、①、②，这时的打印值为：23

①、③、②、④，这时的打印值为：46

①、③、④、②，这时的打印值为：46

③、①、②、④，这时的打印值为：23

③、①、④、②，这时的打印值为：23

由于进程 P1 和 P2 并发执行，共享了变量 K，故产生了‘结果不唯一’。

11 . 证明信号量与管程的功能是等价的：

i. 用信号量实现管程；

ii. 用管程实现信号量。

答：(1) 用信号量实现管程；

Hoare 是用信号量实现管程的一个例子，详见课文内容。下面介绍另一种简单方法：

每一个管程都对应一个 mutex，其初值为 1，用来控制进程互斥调用管程。再设一个初值为 0 的信号量，用来阻塞等待资源的进程。相应的用信号量实现的管程库过程为：

```
    semaphore mutex,c;
    mutex=1;c=0;
procedure enter-monitor( )  {              /*进入管程代码，保证互斥*/
    P(mutex);
}
procedure leave-monitor-normally( )    { /*不发信号退出管程*/
    V(mutex);
}
procedure leave-with-signal(c) {     /*在条件 c 上发信号并退出管程，释放一个等待 c 条
                                     件的进程。 注意这时没有开放管程，因为刚刚被释放
                                     的进程已在管程中。*/
     V(c)
    }
procedure wait(c)  {                  /*等待条件 c，开放管程*/
    V(mutex);
    P(c);
}
```

（2）用管程实现信号量。

```
type semaphore= MONTOR {
   cond S;S=0;
   int C,S_count;C=初值;
InterfaceModule IM;
DEFINE P,V;
USE enter, leave, wait, signal;
  procedure P( ) {
        enter(IM);
        C--;
        if(C<0) wait(S,S_count,IM);
        leave(IM);
      }
procedure V( ) {
        enter(IM);
        C++;
        if(C<=0) signal(S,S_count,IM);
        leave(IM);
      }
```

12. 证明消息传递与管程的功能是等价的：

(1) 用消息传递实现管程；

(2) 用管程实现消息传递。

答：(1) 用消息传递实现管程；

用消息传递可以实现信号量(见 13(2))，用信号量可以实现管程(见 11(1))，那么，把两种方法结合起来，就可以用用消息传递实现管程。

(2) 用管程实现消息传递。

```
TYPE monitor=mailbox {
        int r,k,count，full_count,empty_count;
        message buffer[n];
        cond full,empty; full=empty=0;
        r=0;k=0;count= full_count=empty_count=0;
    InterfaceModule IM;
    DEFINE add,get;
    USE enter,wait,signal,leave;
    procedure add(message r) {
            enter(IM);
            if (count==n ) wait(full,full_count,IM);
            buffer[r]=message;
            r=(r+1) % n
            count++;
            if (count==1) signal(empty,empty_count,IM);
            leave(IM);
        }
    procedure get(message m) {
            enter(IM);
            if(count==0)    wait(empty,empty_count,IM);
            m=buffer[k];
            count--;
            if(count==n-1) signal(full,full_count,IM);
            leave(IM);
        }
```

13. 证明信号量与消息传递是等价的：

iii. 用信号量实现消息传递；

iv. 用消息传递实现信号量。

答：(1) 用信号量实现消息传递；

1) 把消息队列组织成一个共享队列，用一个互斥信号量管理对该队列的入队操作和出队操作。
2) 发送消息是一个入队操作，当队列存储区满时，设计一个同步信号量阻塞 send 操作。
3) 接收消息是一个出队操作，当队列存储区空时，设计另一个同步信号量阻塞 receive 操作。

(2) 用消息传递实现信号量。

1) 为每一个信号量建立一个同步管理进程，它包含了一个计数器，记录信号量值；还为此信号量设立一个等待进程队列。

2) 应用进程执行 P 或 V 操作时，将会调用相应 P、V 库过程。库过程的功能是：把应用进程封锁起来，所执行的 P、V 操作的信息组织成消息，执行 send 发送给与信号量对应的同步管理进程，之后，再执行 receive 操作以接收同步管理进程的应答。
3) 当消息到达后，同步管理进程计数并查看信号量状态。如果信号量的值为负的话，执行 P 操作的应用进程被阻塞，挂到等待进程队列，所以，不再要送回答消息。此后，当 V 操作执行完后，同步管理进程将从信号量相应队列中选取一个进程唤醒，并回送一个应答消息。正常情况下，同步管理进程回送一个空应答消息，然后，解锁执行 P、V 操作的应用程序。

14. 试利用记录型信号量和 PV 操作写出一个不会出现死锁的五个哲学家就餐问题的算法。

答(1)：

```
   semaphore fork[5];
   for(int i=0;i<5;i++)    fork[i]=1;
cobegin
 process philosopher-i( ) { /* i=0,1,2,3 */
      while (true) {
        /*思考*/;
        P(fork[i]);
        P(fork[(i+1)%5]);
        /*吃通心面*/;
        V(fork[i]);
        V(fork[(i+1)%5]);
      }
```

```
process philosopher-i( ){/*i=4 */
     while (true) {
        /*思考*/;
       P(fork[(i+1)%5]);
       P(fork[i]);
       /*吃通心面*/;
       V(fork[i]);
       V(fork[(i+1)%5]);
      }
```

```
coend
```

答(2)：

```
   semaphore fork[5];mutex=1;
   for(int i=0;i<5;i++)    fork[i]=1;
cobegin
 process philosopher-i( )    {    /* i=0,1,2,3,4 */
      while (true) {
        /*思考*/;
       P(mutex);
        P(fork[i]);
        P(fork[(i+1)%5]);
        V(mutex);
        /*吃通心面*/;
        V(fork[i]);
        V(fork[(i+1)%5]);
      }
coend
```

15. Dijkstra 临界区软件算法描述如下：

```
enum {idle,wantin,incs} flag[n] ;
```

```
int turn;
turn=0 or 1 … or n-1;
process Pi( ) {     //i=0,1,…,n-1
   int j;
   do {
     flag [i] =wantin;
     while(turn!=i)
     if(flag [turn] ==idle) turn=i;
     flag [i] =incs;
     j=0;
     while(j<n&&(j==i‖flag [j] !=incs))
       j++;
     }while(j<n);
    {critical section};
   flag [i] =idle;
  }
```

试说明该算法满足临界区管理原则。

答：为方便描述，把 Dijkstra 程序的语句进行编号：

```
       do {
     flag [i] =wantin;                          ①
     while(turn!=i)                             ②
     if(flag [turn] ==idle) turn=i;             ③
     flag [i] =incs;                            ④
     j=0;
     while(j<n&&(j==i‖flag [j] !=incs))         ⑤
       j++;                                     ⑥
     }while(j<n);
    {critical section};
    flag [i] =idle;                             ⑦
  }
     flag [i] =wantin;
     if(flag [turn] ==idle) turn=i;
     flag [i] =incs;
     }while(turn!=i)
     j=0;
     while(j<n && j==i‖flag [j] !=incs)
       j=j+1;
    }while(j>=n);
   {critical section};
   flag [i] =idle;
```

(1) 满足互斥条件

当所有的 Pj 都不在临界区中，满足 flag[j]!=incs (对于所有 j，j!=i)条件时，Pi 才能进入它的临界区，而且进程 Pi 不会改变除自己外的其他进程所对应的 flag[j]的值。另外，进程 Pi 总是先置自己的 flag[i]为 incs 后，才去判别 Pj 进程的 flag[j]的值是否等于 incs，

所以，此算法能保证 n 个进程互斥地进入临界区。

(2) 不会发生无休止等待进入临界区

由于任何一个进程 Pi 在执行进入临界区代码时先执行语句①，其相应的 flag[i]的值不会是 idle。注意到 flag[i]=incs 并不意味着 turn 的值一定等于 i。我们来看以下情况，不失一般性，令 turn 的初值为 0，且 P0 不工作，所以，flag[turn]=flag[0]=idle。但是若干个其他进程是可能同时交替执行的，假设让进程 Pj(j=1,2,..,n-1)交错执行语句①后(这时 flag[j]=wantin)，再做语句②，来查询 flag[turn]的状态。显然，都满足 turn!=i，所以，都可以执行语句③，让自己的 turn 为 j。但 turn 仅有一个值，该值为最后一个执行此赋值语句的进程号，设为 k、即 turn=k(1<=k<=n-1)。接着，进程 Pj(j=1,2,..,n-1)交错执行语句④，于是最多同时可能有 n-1 个进程处于 incs 状态，但不要忘了仅有一个进程能成功执行语句④，将 turn 置为自己的值。

假设{P1,P2,…Pm}是一个已将 flag[i]置为 incs(i=1,2,…,m)(m≤n-1)的进程集合，并且已经假设当前 turn=k(1<=k<=m)，则 Pk 必将在有限时间内首先进入临界区。因为集合中除了 Pk 之外的所有其他进程终将从它们执行的语句⑤(第二个 while 循环语句)退出，且这时的 j 值必小于 n，故内嵌 until 起作用，返回到起始语句①重新执行，再次置 flag[i]=wantin，继续第二轮循环，这时的情况不同了，flag[turn]=flag[k]必定≠idle(而为 incs)。而进程 Pk 发现最终除自身外的所有进程 Pj 的 flag[j]!=incs，并据此可进入其临界区。

16. 一个经典同步问题：吸烟者问题(patil，1971)。三个吸烟者在一个房间内，还有一个香烟供应者。为了制造并抽掉香烟，每个吸烟者需要三样东西：烟草、纸和火柴，供应者有丰富货物提供。三个吸烟者中，第一个有自己的烟草，第二个有自己的纸和第三个有自己的火柴。供应者随机地将两样东西放在桌子上，允许一个吸烟者进行对健康不利的吸烟。当吸烟者完成吸烟后唤醒供应者，供应者再把两样东西放在桌子上，唤醒另一个吸烟者。试采用：(1)信号量和 P、V 操作，(2)管程编写他们同步工作的程序。

答：(1)用信号量和 P、V 操作。

```
    semaphore S,S1,S2,S3;
    S=1;S1=S2=S3=0;
    bool flag1,flag2,flag3;
    flag1=flag2=flag3=true;
cobegin
 process 供应者( ) {
          while(true) {
          P(S);
          /*取两样香烟原料放桌上，由 flagi 标记*/; /*flage1、flage2、flage3 代表烟草、纸、火柴*/
          if(flag2&&flag3)   V(S1);                  /*供纸和火柴*/
            else   if( flag1&&flag3)   V(S2);        /*供烟草和火柴*/
                 else   V(S3);                       /*供烟草和纸*/
          }
    }
 process 吸烟者 1( ) {
          while(true) {
           P(S1);
```

```
            /*取原料;做香烟*/;
            V(S);
            /*吸香烟*/;
          }
      }
process  吸烟者 2( ) {
           while(true) {
            P(S2);
            /*取原料;做香烟*/;
            V(S);
            /*吸香烟*/;
          }
      }
process  吸烟者 3( ) {
            while(true) {
            P(S3);
            /*取原料;做香烟*/;
            V(S);
            /*吸香烟*/;
          }
     }
coend
```

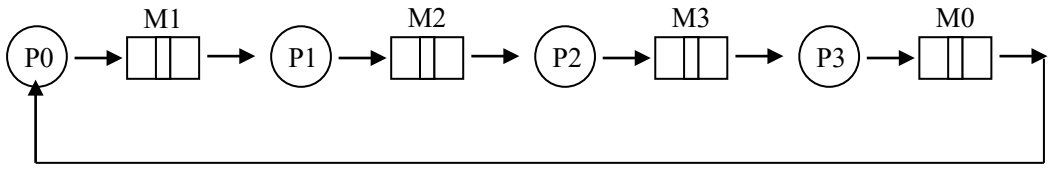
a) 用管程。

```
TYPE   makesmoke=monitor {
          cond S,S1,S2,S3; S=S1=S2=S3=0;
          int S_count,S1_count,S2_count,S3_count;
          S_count=S1_count=S2_count=S3_count=0;
          bool flag1,flag2,flag3;
          flag1=flag2=flag3=true;
InterfaceModule IM;
DEFINE give, take1, take2, take3;
USE enter,wait,signal,leave;
procedure give( ) {
           enter(IM);
           /*准备香烟原料*/;
           if (/*桌上有香烟原料*/)   wait(S,S_count,IM);
           /*把准备的香烟原料放桌上;置对应 flag 为 true*/;
           if(flag2&&flag3)   signal(S1,S1_count,IM);
           else if(flag1&&flag3)   signal(S2,S2_count,IM);
           else   signal(S3,S3_count,IM);
           leave(IM);
      }
 procedure take1( ) {
           enter(IM):
```

```
            if(/*桌上没有香烟原料*/)   wait(S1,S1_count,IM);
              else /*取原料;置对应 flag 为 false*/;
            signal(S,S_count,IM);
            leave(IM);
        }
  procedure take2 ( ) {
            enter(IM):
            if(/*桌上没有香烟原料*/)   wait(S2,S2_count,IM);
              else   /*取原料;置对应 flag 为 false*/;
            signal(S,S_count,IM);
            leave(IM);
        }
  procedure take3 ( ) {
            enter(IM):
            if (/*桌上没有香烟原料*/)   wait(S3,S3_count,IM);
              else   /*取原料;置对应 flag 为 false*/;
            signal(S,S_count,IM);
            release(IM);
        }
cobegin
    process  供应者( ) {
          while(true) {
          makesmoke.give();
          …
         }
        }
    process  吸烟者 1 ( ) {
         while(true) {
           makesmoke.take1();
           /*做香烟，吸香烟*/;
         }
    }
    process  吸烟者 2( ) {
          while(true) {
           makesmoke.take2();
           ./*做香烟，吸香烟*/;
         }
     }
    process  吸烟者 3 ( ){
          while(true) {
           makesmoke.take3();
           /*做香烟，吸香烟*/;
         }
     }
```

coend.

17．四个进程 Pi（i=0…3）和四个信箱 Mj（j=0…3），进程间借助相邻信箱传递消息，即 Pi 每次从 Mi 中取一条消息，经加工后送入 M(i+1)mod4，其中 M0、M1、M2、M3 分别可存放 3、3、2、2 个消息。初始状态下，M0 装了三条消息，其余为空。试以 P、V 操作为工具，写出 Pi（i=0…3）的同步工作算法。



答：

```
    semaphore mutex1,mutex2,mutex3,mutex0;
    mutex1=mutex2=mutex3=mutex0=1;
    semaphore empty0,empty1,empty2,empty3;
    empty0=0;empty1=3;empty2=2;empty3=2;
    semaphore full0,full1,full2,full3;
    full0=3;full1=full2=full3=0;
    int in0,in1,in2,in3,out0,out1,out2,out3;
    in0=in1=in2=in3=out0=out1=out2=out3=0;
cobegin
process P0( ) {
        while(true) {
          P(full0);
          P(mutex0);
          /*从 M0[out0]取一条消息*/;
          out0=(out0+1) % 3;
          V(mutex0);
          V(empty0);
          /*加工消息*/;
          P(empty1);
          P(mutex1);
          /*消息存 M1[in1]*/;
          in1=(in1+1) % 3;
          V(mutex1);
          V(full1);
          }
      }
process P1( ) {
          while(true) {
          P(full1);
          P(mutex1);
          /*从 M1[out1]取一条消息*/;
```

```
            out1=(out1+1) % 3;
            V(mutex1);
            V(empty1);
            /*加工消息*/;
            P(empty2);
            P(mutex2);
            /*消息存 M2[in2]*/;
            in2=(in2+1) % 2;
            V(mutex2);
            V(full2);
            }
          }
process P2( ) {
           while(true) {
            P(full2);
            P(mutex2);
            /*从 M2[out2]取一条消息*/;
            out2=(out2+1) % 2;
            V(mutex2);
            V(empty2);
            /*加工消息*/;
            P(empty3);
            P(mutex3);
            /*消息存 M3[in3]*/;
            in3=(in3+1) % 2;
            V(mutex3);
            V(full3);
            }
          }
process P3( ) {
           while(true) {
            P(full3);
            P(mutex3);
            /*从 M3[out3]取一条消息*/;
            out3=(out3+1) % 2;
            V(mutex3);
            V(empty3);
            /*加工消息*/;
            P(empty0);
            P(mutex0);
            /*消息存 M0[in0]*/;
            in0=(in0+1) % 3;
            V(mutex0);
            V(full0);
```

```
        }
    }
coend
```

18．系统有同类资源 m 个，被 n 个进程共享，问：当 m＞n 和 m≤n 时，每个进程最多可以请求多少个这类资源时，使系统一定不会发生死锁?

答：当 m≤n 时，每个进程最多请求 1 个这类资源时，系统一定不会发生死锁。当 m>n 时，如果 m/n 不整除，每个进程最多可以请求”商+1”个这类资源，否则为”商”个资源，使系统一定不会发生死锁?

19．N 个进程共享 M 个资源，每个进程一次只能申请/释放一个资源，每个进程最多需要 M 个资源，所有进程总共的资源需求少于 M+N 个，证明该系统此时不会产生死锁。

答 1：设 max （i）表示第 i 个进程的最大资源需求量，need(i)表示第 i 个进程还需要的资源量，alloc(i)表示第 i 个进程已分配的资源量。由题中所给条件可知：

max(1)+---+max(n)=(need(1)+---+need(n))+((alloc(1)+---+alloc(n))<m+n

如果在这个系统中发生了死锁，那么一方面 m 个资源应该全部分配出去，

alloc(1)+ ---+alloc(n)=m

另一方面所有进程将陷入无限等待状态。可以推出

need(1)+ ---+need(n)<n

上式表示死锁发生后，n 个进程还需要的资源量之和小于 n，这意味着此刻至少存在一个进程 i,need(i)=0,即它已获得了所需要的全部资源。既然该进程已获得了它所需要的全部资源，那么它就能执行完成并释放它占有的资源，这与前面的假设矛盾，从而证明在这个系统中不可能发生死锁。

答 2：由题意知道，　n×m<m+n　　是成立的，

等式变换　　n×(m-1)+n<n+m

即　　n×(m-1)<m

于是有　　n×(m-1)+1<m+1

或　　n×(m-1)+1≤m

这说明当 n 个进程都取得了最大数减 1 个即(m-1)个时，这时至少系统还有一个资源可分配。故该系统是死锁无关的。

20．一条公路两次横跨运河，两个运河桥相距 100 米，均带有闸门，以供船只通过运河桥。运河和公路的交通均是单方向的。运河上的运输由驳船担负。在一驳船接近吊桥 A 时就拉汽笛警告，若桥上无车辆，吊桥就吊起，直到驳船尾 P 通过此桥为止。对吊桥 B 也按同样次序处理。一般典型的驳船长度为 200 米，当它在河上航行时是否会产生死锁?若会，说明理由，请提出一个防止死锁的办法，并用信号量来实现驳船的同步。

答：当汽车或驳船未同时到达桥 A 时，以任何次序前进不会产生死锁。但假设汽车驶过了桥 A，它在继续前进，并且在驶过桥 B 之前，此时有驳船并快速地通过了桥 A，驳船头到达桥 B，这时会发生死锁。因为若吊起吊桥 B 让驳船通过，则汽车无法通过桥 B；若不吊起吊桥 B 让汽车通过，则驳船无法通过桥 B。可用两个信号量同步车、船通过两座桥的动作。

```
semaphore ship_mutex=1;
```

```
    semaphore car_mutex=0;
    semaphore sbridge=1;
    int ship_count=0;
    in tar_count=0;
    cobegin
     process ship( )   {
            P(ship_mutex);
            if (ship_count==0)   p(sbridge);
            ship_count++;
            V(ship_mutex);
            ship_go_on( );
            P(ship_mutex);
           ship_count--;
        if( ship_count==0)   V(sbridge);
          V(car_mutex);
     }
     process car( )   {
            P(car_mutex);
            if (car_count==0)   p(sbridge);
            car_count++;
            V(ship_mutex);
            car_go_on( );
           P(car_mutex);
           car_count--;
         if(car_count==0)   V(sbridge);
         V(ship_mutex);
     }
coend
```

21. Jurassic 公园有一个恐龙博物馆和一个花园，有 m 个旅客和 n 辆车，每辆车仅能乘一个旅客。旅客在博物馆逛了一会，然后，排队乘坐旅行车，当一辆车可用时，它载入一个旅客，再绕花园行驶任意长的时间。若 n 辆车都已被旅客乘坐游玩，则想坐车的旅客需要等待。如果一辆车已经空闲，但没有游玩的旅客了，那么，车辆要等待。试用信号量和 P、V 操作同步 m 个旅客和 n 辆车子。

答：这是一个汇合机制，有两类进程：顾客进程和车辆进程，需要进行汇合、即顾客要坐进车辆后才能游玩，开始时让车辆进程进入等待状态。

```
    semaphore scl,sck,sc,kx,xc,mutex;
    sck = kx= sc= xc=0;
    sc1=n;mutex=1;
    sharearea:                  /*一个登记车辆\被服务乘客信息的共享区*/
cobegin
process  顾客(i ) {           /*(i=1,2,…)*/
    P(scl);                    /*车辆最大数量信号量*/
    P(mutex);                  /*封锁共享区，互斥操作*/
```

```
    /*在共享区 sharearea 登记被服务的顾客的信息：起始和到达地点，行驶时间*/;
    V(sck);                 /*释放一辆车，即顾客找到一辆空车*/
    P(kx);                  /*车辆要配备驾驶员，顾客等待被载*/
    {上车}；
    V(sc);                  /*顾客进程已汇合到车辆进程，即顾客坐进车里*/
    P(xc);                  /*待游玩结束后后，顾客等待下车*/
    V(scl);                 /*空车辆数加 1*/
  }
process  车辆(j)   {        /*(j=1,2,…)*/
    while(true) {
     P(sck);                /*车辆等待有顾客来使用*/
    /*在共享区 sharearea 登记那一辆车被使用，并与顾客进程汇合*/；
    V(mutex);               /*这时可开放共享区，让另一顾客雇车*/
    V(kx);                  /*允许顾客用此车辆*/
    P(sc);                  /*车辆等待顾客上车*/
    /*车辆载着顾客开行到目的地*/；
    v(xc);                  /*允许顾客下车*/
    }
  }
coend
```

22. 今有 k 个进程，它们的标号依次为 1、2、…、k，如果允许它们同时读文件 file，但必须满足条件：参加同时读文件的进程的标号之和需小于 M（k<M），请使用：1)信号量与 P、V 操作，2)管程，编写出协调多进程读文件的程序。

答 1：1) 使用信号量与 P、V 操作

```
  semaphore waits,mutex;
    int numbersum=0;
    waits=0;mutex=1;
cobegin
     process readeri(int number) {       //i=1,2,…
              P(mutex);
               while(numbersum+number>=M)
                        {V(mutex);P(waits);}
               numbersum=numbersum+number;
              V(mutex);
              Read file;
              P(mutex);
              numbersum=numbersum-number;
              V(waits);
              V(mutex);
     }
coend
```

2) 使用管程：

```
TYPE    sharefile =MONITOR
          int numbersum,n,SF_count; SF_count=0;
          cond SF;SF=0;
          numbersum=0;
InterfaceModule IM
DEFINE startread,endread;
USE wait,signal,enter,leave;
procedure startread(int number) {
          enter(IM);
           while(number+numbersum>=M)    {wait(SF,SF_count,IM);}
           numbersum=numbersum+number;
          leave(IM);
      }
procedure endread(int number) {
              enter(IM);
             numbersum=numbersum-number;
             signal(SF,SF_count,IM);
           leave(IM);
      }
cobegin
       process-i ( );
coend

process-i ( ) {
      int number;
       number=进程读文件编号;
       sharefile.startread(number);
           read F;
       sharefile.endread(number);
   }
```

23．设当前的系统状态如下，系统此时 Available=(1，1，2)：

	Claim			Allocation		
进程,	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0
P2	6	1	3	5	1	1
P3	3	1	4	2	1	1
P4	4	2	2	0	0	2

1)计算各个进程还需要的资源数 Cki-Aki?

2)系统是否处于安全状态，为什么?

3)P2 发出请求向量 request2(1，0，1)，系统能把资源分给它吗？

4)若在 P2 申请资源后，若 P1 发出请求向量 request1(1，0，1)，系统能把资源分给它吗？

5)若在 P1 申请资源后，若 P3 发出请求向量 request3(0，0，1)，系统能把资源分给它吗？

答：1) P1，P2，P3，P4 的 Cki-Aki 分别为：(2，2，2)、(1，0，2)、(1，0，3)、(4，2，0)

2)系统处于安全状态，存在安全序：P2，P1，P3，P4

3)可以分配，存在安全序列：P2，P1，P3，P4。

4)不可以分配，资源不足。

5)不可以分配，不安全状态。

24. 系统有 A、B、C、D 共 4 种资源，在某时刻进程 P0、P1、P2、P3 和 P4 对资源的占有和需求情况如表，试解答下列问题：

Process	Allocation				Claim				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	3	2	0	0	4	4	1	6	2	2
P_1	1	0	0	0	2	7	5	0				
P_2	1	3	5	4	3	6	10	10				
P_3	0	3	3	2	0	9	8	4				
P_4	0	0	1	4	0	6	6	10				

1)系统此时处于安全状态吗？

2)若此时 P2 发出 request2(1、2、2、2)，系统能分配资源给它吗？为什么？

答：(1)系统处于安全状态，存在安全序列：P0，P3，P4，P1，P2。

(2)不能分配，否则系统会处于不安全状态。

25 . 把死锁检测算法用于下面的数据，并请问：

Available=(1，0，2，0)

$$\text{Need}=\begin{pmatrix}1&1&0&0\\0&1&1&2\\3&1&0&0\\0&0&1&0\\2&1&1&0\end{pmatrix}\qquad \text{Allocation}=\begin{pmatrix}3&0&1&1\\0&1&0&0\\1&1&1&0\\1&1&0&1\\0&0&0&0\end{pmatrix}$$

(1) 此时系统处于安全状态吗？

(2) 若第二个进程提出资源请求 request2(0，0，1，0)，系统能分配资源给它吗？

(3) 执行(2)之后，若第五个进程提出资源请求 request5(0，0，1，0)，系统能分配资源给它吗？

答：(1)此时可以找出进程安全序列：P4，P1，P5，P2，P3。故系统处于安全状态。

(2)可以分配，存在安全序列：P4，P1，P5，P2，P3。

(3)不可分配，系统进入不安全状态。

26. 考虑一个共有 150 个存储单元的系统，如下分配给三个进程，P1 最大需求 70,已占有 25；P2 最大需求 60,已占有 40；P3 最大需求 60,已占有 45。使用银行家算法，以确定下面的任何一个请求是否安全。(1)P4 进程到达，P4 最大需求 60,最初请求 25 个。(2)P4 进程到达，P4 最大需求 60,最初请求 35。如果安全，找出安全序列；如果不安全，给出结果分配情况。

答：

1. 由于系统目前还有 150-25-40-45=40 个单元，P4 进程到达，把 25 个单元分给它。这时系统还余 15 个单元，可把 15 个单元分给 P3，它执行完后会释放 60 个单元。于是可供 P1(还要 45 个单元)，P2(还要 20 个单元)，P4(还要 35 个单元)任何一个执行。安全序列为：

   P1，P2，P3，P4，P3，P1，P2，P4
   P1，P2，P3，P4，P3，P1，P4，P2
   P1，P2，P3，P4，P3，P2，P1，P4
   P1，P2，P3，P4，P3，P2，P4，P1
   P1，P2，P3，P4，P3，P4，P1，P2
   P1，P2，P3，P4，P3，P4，P2，P1

2. P4 进程到达，P4 最大需求 60,最初请求 35。如果把 35 个单元分给 P4，系统还余 5 个单元，不再能满足任何一个进程的需求，系统进入不安全状态。

27 . 有一个仓库，可存放 X、Y 两种产品，仓库的存储空间足够大，但要求：(1) 每次只能存入一种产品 X 或 Y， (2) 满足-N<X 产品数量 - Y 产品数量<M。其中，N 和 M 是正整数，试用信号量与 P、V 操作实现产品 X 与 Y 的入库过程。

答：本题给出的表达式可分解为制约条件：

－N＜X 产品数量－Y 产品数量

X 产品数量－Y 产品数量＜M

也就是说，X 产品的数量不能比 Y 产品的数量少 N 个以上，X 产品的数量不能比 Y 产品的数量多 M 个以上。可以设置两个信号量来控制 X、Y 产品的存放数量：

sx 表示当前允许 X 产品比 Y 产品多入库的数量,即在当前库存量和 Y 产品不入库的情况下，还可以允许 sx 个 X 产品入库；初始时，若不放 Y 而仅放 X 产品，则 sx 最多为 M－1 个。

sy 表示当前允许 Y 产品比 X 产品多入库的数量,即在当前库存量和 X 产品不入库的情况下，还可以允许 sy 个 Y 产品入库。初始时，若不放 X 而仅放 Y 产品，则 sy 最多为 N－1 个。

当往库中存放入一个 X 产品时，则允许存入 Y 产品的数量也增加 1，故信号量 sy 应加 1；当往库中存放入一个 Y 产品时，则允许存入 X 产品的数量也增加 1，故信号量 sx 应加 1。

```
    semaphore mutex=1;              /*互斥信号量*/
    semaphore sx，sy；
    sx=M-1; sy=N-1;
cobegin
  process storeX( )  {
       while(true) {
         P(sx);
         P(mutex);
         /*将 X 产品入库*/；
         V(mutex);
```

```
            V(sy);
            }
        }
    process   storeY( )   {
            while(true) {
            P(sy);
            P(mutex);
            /*将 Y 产品入库*/;
            V(mutex);
            V(sx);
            }
        }
coend
```

28. 有一个仓库可存放 A、B 两种零件，最大库容量各为 m 个。生产车间不断地取 A 和 B 进行装配，每次各取一个。为避免零件锈蚀，按先入库者先出库的原则。有两组供应商分别不断地供应 A 和 B，每次一个。为保证配套和合理库存，当某种零件比另一种零件超过 n(n<m)个时，暂停对数量大的零件的进货，集中补充数量少的零件。试用信号量与 P、V 操作正确地实现它们之间的同步关系。

答：按照题意，应满足以下控制关系：A 零件数量- B 零件数量<=n；B 零件数量- A 零件数量<=n；A 零件数量<=m；B 零件数量<=m。四个控制关系分别用信号量 sa、sb、empty1 和 empty2 实施。为遵循先入库者先出库的原则，A、B 零件可以组织成两个循形队列，并增加入库指针 in1、in2 和出库指针 out1、out2 来控制顺序。并发程序编制如下：

```
    semaphore empty1，empty2，full1，full2；
    semaphore mutex，sa，sb；
    int in1，in2，out1，out2；
    item buffer1[m]，buffer2[m] ；
    empty1=empty2=m；sa=sb=n；in1=in2=out1=out2=0；
cobegin
process producerA( ) {
    while(true) {
        P(empty1)；
        P(sa)；
        P(mutex)；
        buffer1[in1]=A 零件；
        in1=(in1+1) % m；
        V(mutex)；
        V(sb)；
        V(full1)；
        }
     }
process producerB( ) {
     while(true) {
        P(empty2)；
```

```
            P(sb)；
            P(mutex)；
            buffer2[in2]=B 零件；
            in2=(in2+1) % m；
            V(mutex)；
            V(sa)；
            V(full2)；
            }
        }
process take( ) {
          while(true) {
        P(full1)；
        P(full2)；
        P(mutex)；
        {take from buffer1[out1] and buffer2[out2]中的 A、B 零件}；
        out1=(out1+1) % m；
        out2=(out2+1) % m；
        V(mutex)；
        V(empty1)；
        V(empty2)；
        把 A 和 B 装配成产品；
        }
      }
coend.
```

29. 进程 A1、A2、…、An1 通过 m 个缓冲区向进程 B1、B2、…、Bn2 不断地发送消息。发送和接收工作符合以下规则：

(1) 每个发送进程每次发送一个消息，写进一个缓冲区，缓冲区大小与消息长度相等；

(2) 对每个消息，B1、B2、…、Bn2 都需接收一次，并读入各自的数据区内；

3. 当 M 个缓冲区都满时，则发送进程等待，当没有消息可读时，接收进程等待。

试用信号量和 PV 操作编制正确控制消息的发送和接收的程序。

答：本题是生产者－消费者问题的一个变形，一组生产者 A1，A2，…An1 和一组消费者 B1，B2，…Bn2 共用 m 个缓冲区，每个缓冲区只要写一次，但需要读 n2 次。因此，可以把这一组缓冲区看成 n2 组缓冲区，每个发送者需要同时写 n2 组缓冲区中相应的 n2 个缓冲区，而每一个接收者只需读它自己对应的那组缓冲区中的对应单元。

应设置一个信号量 mutex 实现诸进程对缓冲区的互斥访问；两个信号量数组 empty[n2] 和 full[n2]描述 n2 组缓冲区的使用情况。其同步关系描述如下：

```
semaphore mutex,empty[n2],full[n2];
int i;mutex=1;
for(i=0;i<n2;i++) {
   empty[i]=m; full[i]=0;
   }
   main( ) {
   cobegin
```

```
      A1( );
      A2( );
      ┇
     An1 ();
     B1 ();
     B2 ();
      ┇
     Bn2( );
   coend
    }

procedure send (   )    {          /*进程 Ai 发送消息*/
   for (int i=0;i<n2;i++)
         p(empty[i]);
         p(mutex);
         /*将消息放入缓冲区*/
         V（mutex）;
   for(int i=0;i<n2;i++)
         V(full[i]);
   }
procedure receive(i)   {   /*进程 Bi 接收消息*/
      p(full[i]);
      p(mutex);
      /*将消息从缓冲区取出*/
      V(mutex);
      V(emtpy[i]);
}
Ai (   )   {           /*发送进程 A1,A2,…An1 的程序类似，这里给出进程 Ai 的描述*/
while (true) {
    send (   );
      ┇
    }
}
Bi (   )   {        /*接收进程 B1,B2,…Bn2 的程序类似，这里给出进程 Bi 描述*/
while (true)   {
      receive(i);
      ┇
    }
}
```

30 ．某系统有 R1 设备 3 台，R2 设备 4 台，它们被 P1、P2、P3 和 P4 进程共享，且已知 4 个进程均按以下顺序使用设备：

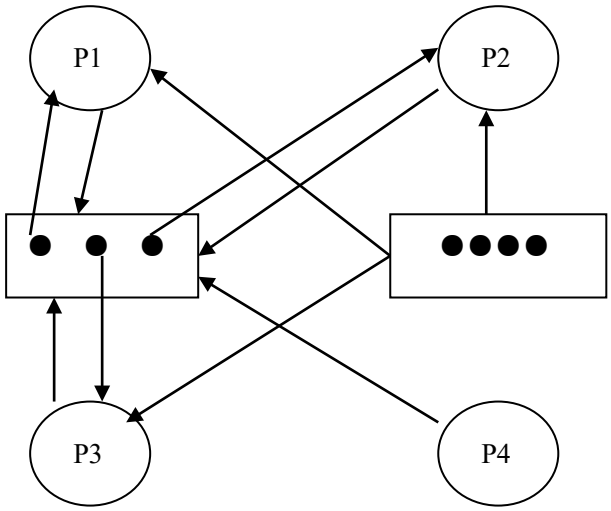
→申请 R1→申请 R2→申请 R1→释放 R1→释放 R2→释放 R1

(1) 系统运行中可能产生死锁吗？为什么？

(2) 若可能的话，请举出一种情况，并画出表示该死锁状态的进程—资源图。

答：(1)系统四个进程需要使用的资源数为 R1 各 2 台，R2 各 1 台。可见资源数不足，同时各进程申请资源在先，有可能产生死锁发生的四个条件，故系统可能产生死锁。

(2) 当三个进程执行完申请资源 R1，开始执行申请资源 R2 时，第四个进程会因没有资源 R1 而被阻塞。当三个进程执行完申请资源 R2 后，系统还剩 1 个 R2 资源。而这三个进程因执行申请第二个资源 R1 而全部被阻塞，系统进入死锁。



31. 独木桥问题 1：东西向汽车过独木桥，为了保证安全，只要桥上无车，则允许一方的汽车过桥，待一方的汽车全部过完后，另一方的汽车才允许过桥。请用信号量和 PV 操作来写出汽车过独木桥问题的同步算法。

答：

```
        semaphore wait,mutex1,mutex2;
        mutex1=mutex2=1;wait=1;
        int count1,count2; count1=0;count2=0;
cobegin
    process P 东( ) {                          process P 西( ) {
          P(mutex1);                                P(mutex2);
          count1++;                                 count2++;
          if (count1==1)    P(wait);                if (count2==1)    P(wait);
          V(mutex1);                                V(mutex2);
          /*过独木桥*/;                               /*过独木桥*/;
          P(mutex1);                                P(mutex2);
          Count1--;                                 count2--;
          If( count1==0)    V(wait);                if (count2==0)    V(wait);
          V(mutex1);                                V(mutex2);
       }                                          }
coend
```

32. 独木桥问题 2：在独木桥问题 1 中，限制桥面上最多可以有 k 辆汽车通过。试用信号量

和 P，V 操作写出汽车过独木桥问题的同步算法。

答 1：

```
    semaphore wait,mutex1,mutex2,bridge;
    mutex1=mutex2=1;bridge=k;wait=1;
    int count1,count2; count1=0;count2=0;
cobegin
      process P 东( ) {                          process P 西( ) {
        P(mutex1);                                 P(mutex2);
        count1++;                                  count2++;
        if (count1==1)   P(wait);                 if (count2==1)   P(wait);
        V(mutex1);                                 V(mutex2);
        P(bridge);                                 P(bridge);
        /*过桥*/                                    /*过桥*/
        V(bridge);                                   V(bridge);
        P(mutex1);                                 P(mutex2);
        count1--;                                    count2--;
        if (count1==0)   V(wait);                 if (count2==0)   V(wait);
        V(mutex1);                                   V(mutex2);
     }                                           }
coend
```

答 2：

```
cobegin
      process P 东( ) {                          process P 西( ) {
        P(bridge1);                                P(bridge2);
        P(mutex1);                                 P(mutex2);
        count1++;                                  count2++;
      if (count1==1 )   P(wait);                 if (count2==1)   P(wait);
        V(mutex1);                                   V(mutex2);
        /*过桥*/                                    /*过桥*/
        V(bridge1);                                V(bridge2);
        P(mutex1);                                   P(mutex2);
        count1--;                                    count2--;
      if (count1==0) V(wait)                     if (count2==0 ) V(wait);
        V(mutex1);                                   V(mutex2);
     }                                           }
coend
```

33．独木桥问题 3：在独木桥问题 1 中，以叁辆汽车为一组，要求保证东方和西方以组为单位交替通过汽车。试用信号量和 P，V 操作写出汽车过独木桥问题的同步算法。

解：

semaphore wait,mutex1,mutex2;
mutex1=mutex2=1;wait=1;
int counter1,counter2; counteru1=0; countd1=0; counteru2=0; counterd2=0;
semaphore S1,S2;S1=3;S2=0;	

cobegin
Process P 左() {
while(true) {
P(S1)
P(mutex1);
countu1++;
	
	Process P 右() {
	while(true) {
	P(S2)
	P(mutex2);
	countu2++;

```
            if (countu1==1) P(wait);
        V(mutex1);
            /*过独木桥*/
        V(S2)
         P(mutex1);
            countu1--;
            countd1++
            if ((countu1==0)&(countd1==3))
                {countd1=0; V(wait); }
         V(mutex1);
    }
}
coend
```

```
            if (countu2==1) P(wait);
        V(mutex2);
            /*过独木桥*/
        V(S1)
        P(mutex2);
           countu2--;
           countd2++
           if ((countu2==0)&(countd2==3))
                {countd2=0; V(wait); }
        V(mutex2);
    }
}
```

34. 独木桥问题 4：在独木桥问题 1 中，要求各方向的汽车串行过桥，但当另一方提出过桥时，应能阻止对方未上桥的后继车辆，待桥面上的汽车过完桥后，另一方的汽车开始过桥。试用信号量和 P，V 操作写出汽车过独木桥问题的同步算法。

解：stop 用于当另一方提出过桥时，应阻止对方未上桥的后继车辆。

```
        semaphore stop,wait,mutex1,mutex2;
        stop=mutex1=mutex2=1;wait=1;
        int count1,count2; count1=0;count2=0;
  cobegin
    process P 东( ) {
        P(stop);
        P(mutex1);
        count1++;
     if (count1==1)    P(wait);
        V(mutex1);
        V(stop);
        /*过桥/*
        P(mutex1);
        Count1--;
    if (count1==0)    V(wait);
        V(mutex1);
    }
coend
```

```
    process P 西( ) {
      P(stop);
      P(mutex2);
       count2++;
    if (count2==1)    P(wait);
     V(mutex2);
     V(stop);
      /*过桥}*/
       P(mutex2);
       count2--;
  if (count2==0)    V(wait);
     V(mutex2);
  }
```

35．假设一个录像厅有 0、1 和 2 三种不同的录像片由观众选择放映。录像厅放映的规则为：(1)任一时刻最多只能放映一种录像片，正在放映的录像片是自动循环放映的，最后一个观众主动离开时结束当前录像片的放映。(2)选择当前正在放映录像片的观众可立即进入，允许同时有多位选择同一录像片的观众同时观看，同时观看数量不受限制。(3)等待观看其他录像片的观众按到达顺序排队，当一种新的录像片开始放映时，所有等待观看该录像片的观众可依次进入录像厅同时观看。用一个进程代表一个观众，实现观众进程观看录像函数 Videoshow(int Vcd_id)，以遵守放映规则。Vcd_id 表示观众选择的录像编号。要求用信号量和 PV 操作写出同步活动的程序。

解：

```
    semaphore wait.S0,S1,S2;
    int count0,count1,count2;
    count0=count1=count2=0;
```

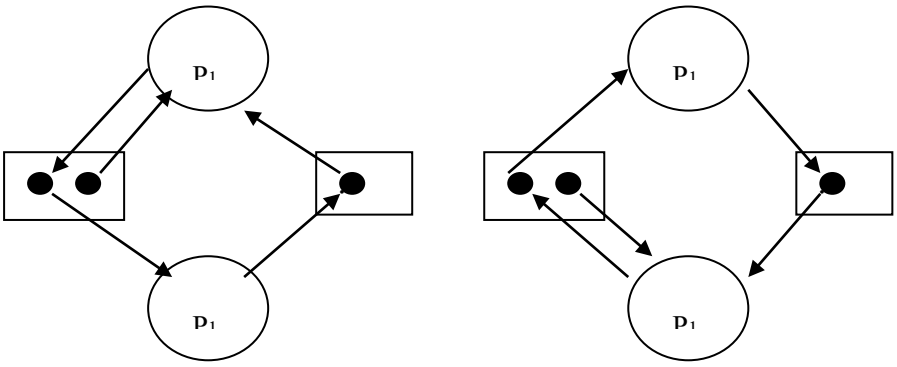
```
    wait=S0=S1=S2=1;
cobegin
  process Videoshow0( )   {        /*观看 0 种录像观众进入录像厅*/
        while(true) {
        P(S0);
        Count0++;
        if (count0==1)   P(wait);   /*没有人看录像，将放 0 号录像片*/
        V(S0);
        /*看录像片 0*/
        P(S0);
        count--;
        if (count0==0)   V(wait); /*最后一个，通知看其他录像片等待队列的观众*/
        V(S0);
        }
     }
  process Videoshow1( ) {       /*观看 1 种录像观众进入录像厅*/
        while(true) {
        P(S1);
        Count1++;
        if (count1==1)   P(wait); /*没有人看录像，将放 1 号录像片*/
        V(S1);
        /*看录像片 1*/
        P(S1);
        count--;
        if (count1==0)   V(wait); /*最后一个，通知看其他录像片等待队列的观众*/
        V(S1);
        }
     }
  process Videoshow2( ) {       /*观看 2 种录像观众进入录像厅
        while(true) {
        P(S2);
        Count2++;
        if (count2==1)   P(wait); /*没有人看录像，将放 2 号录像片*/
        V(S2);
        /*看录像片 2*/
        P(S2);
        count--;
        if (count2==0)   V(wait); /*最后一个，通知看其他录像片等待队列的观众*/
        V(S2);
        }
     }
coend.
```

36．假定某计算机系统有 R1 和 R2 两类可再使用资源（其中 R1 有两个单位，R2 有一个单位），它们被进程 P1，P2 所共享，且已知两个进程均以下列顺序使用两类资源。

→申请 R1→申请 R2→申请 R1→释放 R1→释放 R2→释放 R1→

试求出系统运行过程中可能到达的死锁点，并画出死锁点的资源分配图（进程－资源图）。

答：当两个进程都执行完第一步(都占用 R1) 时，系统进入不安全状态。这时无论哪个进程执行完第二步，死锁都会发生。可能到达的死锁点：进程 P1 占有一个 R1 和一个 R2，而进程 P2 占有一个 R1。或者相反。这时已形成死锁。进程---资源图为：



37. 某工厂有两个生产车间和一个装配车间，两个生产车间分别生产 A、B 两种零件，装配车间的任务是把 A、B 两种零件组装成产品。两个生产车间每生产一个零件后都要分别把它们送到装配车间的货架 F1、F2 上，F1 存放零件 A，F2 存放零件 B，F1 和 F2 的容量均为可以存放 10 个零件。装配工人每次从货架上取一个 A 零件和一个 B 零件，然后组装成产品。请用：(1)信号量和 P、V 操作进行正确管理，(2)管程进行正确管理。

答：(1) 信号量和 P、V 操作进行正确管理。

```
    item F1[10],F2[10];
    semaphore SP1,SP2,SI1,SI2;
    int in1,in2,out1,out2;
    in1=0;in2=0;out1=0;out2=0;
    SP1=10;SP2=10;SI1=0;SI2=0;
    main( ) {
   cobegin
    producer1( ); producer2( ); installer( );
   coend
 }
process producer1() {
     while (true) {
        produce A 零件;
        P(SP1);
        F1[in1]=A;
        in1=(in1+1)%10;
        V(SI1);
       }
   }
process producer2( ) {
        while (true) {
        produce   B 零件;
        P(SP2);
        F2[in2]=B;
```

```
        in2=(in2+1) % 10;
        V(SI2);
         }
    }
process installer() {
   item product;
     while (true) {
          P(SI1);
          product1=F1[out1];
          out1=(out1+1) % 10;
          V(SP1);
          P(SI2);
          product2=F2[out2];
          out2=(out2+1) % 10;
          V(SP2);
          /*组装产品*/
     }
    }
```

(2) 管程进行正确管理。

```
TYPE   produceprodut=monitor
      item F1[10],F2[10];
      cond SP1,SP2,SG1,SG2; SP1=SP2=SG1=SG2=0;
      int SP1_count,SP2_count2,SG1_count,SG2_count;
      SP1_count=SP2_count2=SG1_count=SG2_count=0;
      int in1,in2,out1,out2;
      int inc1,inc2;
      in1=0;in2=0;out1:=0;out2=0;inc1=0;inc2=0;
InterfaceModule IM;
DEFINE put1, put2, get;
USE   wait,signal,enter,leave;
procedure put1(A){
  enter(IM);
  if(inc1==10) wait(SP1,SP1_count,IM);
  inc1++;
  F1[in1]=A;
  in1=(in1+1) % 10;
  signal(SG1,SG1_count, IM);
  leave(IM);
}
procedure put2(B){
   enter(IM);
  if(inc2==10) wait(SP2,SP2_count,IM);
  inc2++;
  F2[in2]=B;
  in2=(in2+1) % 10;
```

```
  signal(SG2,SG2_count, IM);
  leave(IM);
 }
procedure get(A,B) {
   enter(IM);
  if(inc1==0) wait(SG1,SG1_count,IM);
  if(inc2==0) wait(SG2,SG2_count,IM);
  inc1--;
  inc2--;
  A=F1[out1];
  out1:=(out1+1) % 10;
  B=F2[out2];
  out2=(out2+1) % 10;
  signal(SP1,SP1_count,IM);
  signal(SP2,SP2_count,IM);
  leave(IM);
}
cobegin
  process produce1( ) {
       while (true) {
          produce   A 零件;
          produceprodut.put1(A);

       }
  process produce2 {
      while (true) {
         produce 零件;
          produceprodut.put2(B);
       }
     }
  process consumer( ) {
          while (true) {
          produceprodut.get(A,B);
          {组装产品};
         }
     }
  coend.
```

38．桌上有一只盘子，最多可以容纳两个水果，每次仅能放入或取出一个水果。爸爸向盘子中放苹果(apple)，妈妈向盘子中放桔子(orange)，两个儿子专等吃盘子中的桔子，两个女儿专等吃盘子中的苹果。试用：(1)信号量和 P、V 操作，(2)管程，来实现爸爸、妈妈、儿子、女儿间的同步与互斥关系。

答：(1)用信号量和 P、V 操作。
类似于课文中的答案，扩充如下：1) 同步信号量初值为 2；2) 要引进一个互斥信号量 mutex，用于对盘子进行互斥；3)盘子中每一项用橘子、苹果 2 个枚举值。

```
        enum {apple, orange} plate [2];
        boolean flag0,flag1;
        semaphore mutex;
        semaphore sp;              /* 盘子里可以放几个水果 */
        semaphore sg1, sg2;        /* 盘子里有桔子，有苹果?*/
        sp=2;                      /* 盘子里允许放入二个水果*/
        sg1=sg2=0;                 /* 盘子里没有桔子，没有苹果 */
        flag0=flag1=false;mutex=1;
    cobegin
    process father () {                    process son () {
        while(true) {                          while(true) {
        /*削一个苹果*/                          P(sg1);
        P(sp);                                 P(mutex);
        P(mutex);                              if (flag0&&plate[0]= =桔子)
        if (flag0==false)                        { x = plate[0];flag0=false;}
          {plate [0]=苹果; flag0=true;}        else { x= plate[1]; flag1=false;}
        else { plate[1]=苹果; flag1=true; }    V(mutex);
        V(mutex);                              V(sp);
        V(sg2);                                /*吃桔子*/
        }                                      }
     }                                      }
    process mother () {                    process daughter () {
        while(true) {                          while(true) {
        /*剥一个桔子*/                          P(sg2);
        P(sp);                                 P(mutex);
        P(mutex);                              if (flag0&&plate[0]==苹果)
        if (flag0==false)                        { x= plate[0];flag0=false;}
         {plate [0]=桔子; flag0=true;}         else { x= plate[1]; flag1=false;}
       else {plate[1]=桔子; flag1=true; }      V(mutex);
        V(mutex);                              V(sp);
        V(sg1);                                /*吃苹果*/
        }                                      }
    }                                      }
    Coend
```

(2)用管程。

```
TYPE   FMSD = MONITOR
       enum {apple, orange} plate[2];
       int count; bool flag0,flag1;
       cond SP, SS, SD ; SP,=SS,=SD=0;
       int SP_count,SS_count,SD_count; SP_count=SS_count=SD_count=0;
       count=0;flag0=false;flag1=false;
       plate[0]=plate[1]=null;
InterfaceModule IM;
DEFINE put, get;
```

```
USE wait, signal, enter, leave；
procedure put(FRUIT fruit) {
    enter(IM);
    if(count==2)    wait(SP,SP_count,IM)；
    else {if(flag0==false)
                {plate[0]=fruit;flag0=true;}
                  else {plate[1]=fruit; flag1=true; }
                  count++;
                 if(fruit==orange)    signal(SS,SS_count,IM)；
                 else signal(SD,SD_count,IM);
                }
      leave(IM);
  }
procedure get(FRUIT fruit, FRUIT &x) {
     enter(IM);
      if ((count==0) || plate!=fruit)
            if (fruit==orange)    wait(SS,SS_count,IM)；
          else    wait(SD,SD_count,IM)；
          count--;
        if (flag0&&plate[0]= =fruit)
              {x=plate[0];flag0=false;}
        else {x=plate[1]; flag1=false;}
     signal(SP, SP_count,IM);
     leave(IM);
}
 cobegin
    process father( ) {
      while (true) {
        /*准备好苹果*/
        FMSD.put(apple);
       }
     }
    process mother( ) {
      while (true) {
        /*准备好桔子*/
        FMSD.put(orange);
       }
    }
process son( ) {
  while (true) {
      FMSD.get(orange,x);
      /*吃取到的桔子*/
    }
  }
```

```
process daughter( ) {
    while (true) {
        FMSD.get(apple,x);
        /*吃取到的苹果*/
    }
  }
coend
```

39．一组生产者进程和一组消费者进程共享九个缓冲区，每个缓冲区可以存放一个整数。生产者进程每次一次性向 3 个缓冲区写入整数，消费者进程每次从缓冲区取出一个整数。请用：(1)信号量和 P、V 操作，(2)管程，写出能够正确执行的程序。

答：(1)信号量和 P、V 操作。

```
var   int buf[9];
      int count,getptr,putptr;
      count=0;getpt=0;putpt=0;
      semaphoreS1,S2,SPUT,SGET;
      S1=1;S2=1;SPUT=1;SGET=0;
  main( ) {
     cobegin
      producer-i();consumer-j();
     coend
 }
process producer-i( ) {
        while(true) {
         /*生产 3 个整数*/
          P(SPUT);
          P(S1);
          buf[putptr]=整数 1;
          putptr=(putptr+1) % 9;
          buf[putptr]=整数 2;
          putptr=(putptr+1) % 9;
          buf[putptr]=整数 3;
          putptr=(putptr+1) % 9;
          V(SGET);
          V(SGET);
          V(SGET):
          V(S1);
      }
  }
process consumer-j( ) {
        int y;
        while(true) {
         P(SGET);
```

```
            P(S2);
            y=buf[getptr];
            getptr=(getptr+1) % 9;
            count++;
            if (count==3)
                  {count=0;V(SPUT);}
              V(S2);
              {consume the  整数 y};
              }
        }
```

(2) 管程。

```
TYPE    get_put = MONITOR
            int buf [9];
            int count,getptr,putptr;
            cond SP,SG; SP=SG=0;
            int SP_count,SG_count; SP_count=SG_count=0;
          count:=0;getptr:=0;putptr:=0;
InterfaceModule IM;
DEFINE put, get;
USE wait, signal, enter, leave；
procedure put(int a1,int a2,int a3) {
       enter(IM);
     if (count>6)    wait(SP,SP_count,IM)；
     count=count+3;
     buf[putptr]=a1;
     putptr=(putptr+1) % 9;
     buf[putptr]=a2;
     putptr=(putptr+1) % 9;
     buf[putptr]=a3;
     putptr=(putptr+1) % 9;
   signal(SG,SG_count,IM);
   signal(SG,SG_count,IM);
   signal(SG,SG_count,IM);
   leave(IM);
}
procedure get(int b) {
       enter(IM);
         if (count= =0)    wait(SG,SG_count,IM)；
         b=buf[getptr];
         getptr=(getptr+1) % 9;
         count=count--;
         if (count < 7)    signal(SP,SP_count, IM);
         else if (count > 0)    signal(SG,SG_count,IM);
         leave(IM);
```

```
}
cobegin
   process producer-i( ) {
      while(true) {
      {生产 3 个整数};
     get-put.put(a1,a2,a3);
   }
}
process consumer-j( ) {
      while(true) {
      get-put.get(b)
      {consume the  整数 b};
      }
   }
coend
```

40．设计一条机器指令和一种与信号量机制不同的算法，使得并发进程对共享变量的使用不会出现与时间有关的错误。

解：

(1)设计机器指令。

设计一条如下的”测试、比较和交换”三地址指令，提供了一种硬件互斥解决方案：

TC&S	R1	R3	B2	D2

该指令的功能如下：

1) C 为一个共享变量，由地址 2、即变址(B2)+D2 给出，
2) (R1)与(C)比较，
3) 如果 (R1)=(C) 则 (R3)→C，并置条件码为”00”，
   如果 (R1)!=(C) 则 (C)→R1，并置条件码为”01”。

(2)编写进程访问共享变量的程序。

对每个访问共享变量 C 的进程，编写访问共享变量的程序段为：

临界区程序	说　明
(C) →R1； loop2：(R1) →R3； Add/decrease R3 ； TC&S； 「(condition=01)　loop2；	---共享变量 C 的值保护到 R1 中。 --- R1 的值传送到 R3 中，进程修改共享变量时，先对 R3 操作(不是直接操作 C)。 ----R3 加 1/减 1，进程归还/申请由共享变量 C 代表的共享资源(假定每次一个)。 ----执行”测试、比较和交换”指令。 - --条件码=01，转向循环 loop2；否则离开临界区。

(3)程序执行说明。

此解与互斥使用共享变量的思路绝然不同，并发运行的进程可不互斥地访问它们的共享变量。此方案认为造成共享变量 C 值错误的原因在于：一个进程(P1)在改变 C 值的过程中，另一个进程(P2)插进来也改变了 C 的值，而本进程(P1)却不知道，造成了 C 值结果不正确。如果有办法使本进程(P1)能知道 C 值是否改变，改变的话在继承改变了的 C 值的基础上，再作自己的改变操作，则就不会导致共享变量 C 值的错误。

为此，本解决方案中，当一个进程(P1)准备改变 C 值时，先把 C 的值保护在 R1 中，然后，通过 R3 来改变共享变量 C 的值。当要把新的值(即 R3 内的值)送 C 之前，先要判断一下在本进程(P1)工作期间是否有别的进程(P2)插进来也改变了 C 的值(并发进程 P1、P2 的执行完全会造成这种情况)，方法是：将(R1)中被保护的 C 的原来值，与 C 的当前值比较，若相等，说明 C 值未被改变过，则将本进程(P1)修改过的新值送 C(即(R3)→C)；若不相等，说明 C 值在工作期间被改变过，则应该继承 C 的新值(即(C)→R1)并且返回到 loop2 处重新对 C 值计数，以此保证 C 值的最终结果的正确性。

这里提及” 进程工作期间”指的是一个进程从开始至结束对共享变量 C 值的操作的这段时间，也就是执行进程”临界区”这段程序的时间。此外，在进程进入临界区之前，应等待直到 C 为非 0(即有资源可用)为止。

(4)举例。

假定系统中有静态分配资源磁带机共 3 台，被 N 个进程共享，由共享变量 C 来代表可用磁带机台数，其初值为 3。现有并发进程 P1 和 P2 均申请使用磁带机，执行临界区程序。

进程 P1 执行临界区程序

(C) →R1；	因(C)=3，故(R1)=3。
loop2：(R1) →R3；	因(R1)=3，故(R3)当前也=3。
decrease R3 ；	申请使用磁带机，做减 1 操作，故(R3)=2。
TC&S	执行”测试、比较和交换”TC&S 指令。 如果 (R1)=(C) 则 (R3)→C，即(C)=2，并置条件码为”00”，跳出临界区程序，去使用磁带机。 如果 (R1)≠(C) ，例如，(C)=2，说明进程 P2 抢先申请了磁带机，所以，C 与保护在 R1 中的值不一样了(C 的值必小于 R1 的值)，应以 C 的当前值为准，执行(C)→R1(R1 此时变为 2)，并置条件码为”01”，转向 loop2。于是(R1)=2,跟着(R3)=2。接着(R3)减 1 后应=1 了。再执行 TC&S 时，由于(R1)=(C)=2，会使 C 变为 1。
Γ(condition=01) loop2；	

41 ．下述流程是解决两进程互斥访问临界区问题的一种方法。试从“互斥”(mutual exclusion)、“空闲让进”(progress)、“有限等待”(bounded waiting)等三方面讨论它的正确性。如果它是正确的，则证明之；如果它不正确，请说明理由。

```
int c1,c2;
procedure p1( ) {                    procedure p2( ) {
  remain section 1;                  remain section 2;
  do {                               do {
```

```
一          c1=1-c2;                                    c2=1-c1;
        }while(c2==0);                                  }while(c1==0);
        /*critical   section*/                          /* critical   section*/
        c1=1;                                           c2=1;
         }                                              }
         main( ) {                  /*主程序*/
           c1=1; c2=1;
           cobegin
                p1( );p2( );                /*两进程p1, p2并发执行*/
           coend
         }
```

答：(1)互斥

已知c1和c2的初值为1，若进程P1执行到c1=1-c2时，进程P2也同时执行c2=1-c1。这样一来，c1和c2的值都变为0，接着再各自执行循环语句c1=1-c2和c2=1-c1时，c1和c2就又都变回了1。于是，P1和P2会同时进入临界区，不满足互斥条件。

(2) 有空让进

设开始无进程在临界区中，进程P1执行了c1=1-c2，由于c2的初值为1，这使得c1的值变为0但c2仍为1，从而保证了P1进入临界区。当P1退出临界区时，执行了c1=1，使得P2就可进入临界区。进程P2先执行的情况相似，能保证有空让进的原则。

(3) 有限等待

假定进程P1在临界区执行，进程P2申请进入临界区，则因进程P1会在有限时间内执行完并退出临界区，然后，将执行c1=1，这使得进程P2因c1值为1而立即可进入临界区。因而，能满足有限等待的原则。

42．举例说明下列算法不能解决互斥问题。

```
bool blocked[2];
int turn= 0 or 1;
blocked [0] =blocked [1] =false;
turn=0;
procedure P(int id) {
   blocked [id] =true;
   do {
       while(blocked [1-id] ) ;
       turn=id;
   }while(turn!=id);
   {Critical Section};
   blocked [id] =false;
   {remainder}
 }
```

```
cobegin
  P［0］;P［1］;
coend;
```

答：为方便描述，把程序语句进行编号：

```
blocked[id]=true;                ①
 do {
    while(blocked［1-id］) ; ③
    turn=id;                     ④
  }while(turn!=id);              ②
```

假设id=0，则1-id=1，并且turn=1。当进程P[id]先执行①置blocked[id]=true；接着执行②时，因为turn!=id而进入到③执行。此时，因blocked[1-id]为false(初值)，故在③上不做空操作而打算去做④。麻烦的事情发生了，如果在P[id]执行④之前，系统又调度执行P[1-id]，而P[1-id]在执行了①置blocked[1-id]=true之后，在执行②时，因发现turn=1-id，故退出了while，直接进入临界区。而这时P[id]继续执行④，虽然置turn=id但已无法挡住P[1-id]先已进入了临界区的事实，此后，P[id]也进入临界区。

所以，该算法不能解决互斥问题，它会让两个进程同时进入临界区。

43. 现有三个生产者 P1、P2、P3，他们都要生产桔子水，每个生产者都已分别购得两种不同原料，待购得第三种原料后就可配制成桔子水，装瓶出售。有一供应商能源源不断地供应糖、水、桔子精，但每次只拿出一种原料放入容器中供给生产者。当容器中有原料时需要该原料的生产者可取走，当容器空时供应商又可放入一种原料。假定：

生产者 P1 已购得糖和水；

生产者 P2 已购得水和桔子精；

生产者 P3 已购得糖和桔子精；

试用：1)管程，2)信号量与 P、V 操作，写出供应商和三个生产者之间能正确同步的程序。

答：1)管程。

```
TYPE makedrink=monitor
        cond S,S1,S2,S3; S=S1=S2=S3=0;
        int S_count,S1_count,S2_count,S3_count; S_count=S1_count=S2_count=S3_count=0;
        enum {糖,水,桔子精} container;
InterfaceModule IM;
DEFINE give，produce1，produce2，produce3;
USE enter,wait,signal,leave;
procedure give( ) {
          enter(IM);
          take material;
          if (container!=null)   wait(S,S_count,IM);
          else container=material;
          if(container==桔子精)   signal(S1,S1_count,IM);
```

```
            else if(container==糖)    signal(S2,S2_count,IM);
                    else signal(S3,S3_count,IM);
            leave(IM);
        }
procedure produce1( ) {
            enter(IM);
            if (container!=桔子精)    wait(S1,S1_count,IM);
              else {take  桔子精  from container;  做桔子水;}
            signal(S,S_count,IM);
            leave(IM);
        }
procedure produce2( ) {
            enter(IM);
            if (container!=糖)    wait(S2,S2_count,IM);
              else { take  糖 from container;做桔子水;}
            signal(S,S_count,IM);
            leave(IM);
         }
procedure produce3( )    {
            enter(IM);
            if (container!=水)    wait(S3,S3_count,IM);
              else { take  水 from container;  做桔子水;}
            signal(S,S_count,IM);
            leave(IM);
         }
    cobegin
    process 供应商( ) {
      while(true) {
        makedrink.give();
        }
     }
    Process P1( ) {
      while(true) {
          makedrink.produce1();
        }
     }
    Process P2( ) {
      while(true) {
          makedrink.produce2();
          }
     }
    Process P3( ) {
      while(true) {
          makedrink.produce3();
```

```
        }
    }
    coend.
```

2) 信号量与 P、V 操作

```
    semaphore S,S1,S2,S3;
    S=1;S1=S2=S3=0;
    enum container{糖,水,桔子精} ;
  cobegin
   process 供应商( ) {
            while(true) {
             P(S);
             {take material into container};
             if(container==桔子精)   V(S1);
             else   if(container==糖)   V(S2);
                    else   V(S3);
             }
      }
process P1( ) {
          while(true) {
            P(S1);
            {take 桔子精 from container};
            V(S);
            {做桔子水};
            }
    }
process P2( ) {
           while(true) {
            P(S2);
            {take 糖 from container};
            V(S);
            {做桔子水};
           }
    }
process P3( ) {
          while(true) {
            P(S3);
            {take 水 from container};
            V(S);
            {做桔子水};
            }
    }
coend.
```

44．进程A向缓冲区buffer发消息，每当发出一消息后，要等待进程B、C、D都接收这条消息后，进程A才能发新消息。(1)用信号量和PV操作，(2)用管程，写出它们同步工作的程序。

答：(1)用信号量和PV操作。

本质上是一个生产者与三个消费者问题。缓冲区buffer只要写一次，但要读三次。可把buffer看作用三个缓冲块组成的缓冲区，故Sa初值为3。

```
semaphore Sa,Sb,Sc,Sd;
Sa=3;Sb=Sc=Sd=0;
message buffer;
cobegin
process A( )  {
 while(true) {
        P(Sa);
        P(Sa);
        P(Sa);
        /*Send message to buffer*/
        V(Sb);
        V(Sc);
        V(Sd);
       }
  }
process B( )  {
    while(true) {
       P(Sb);
       /*receive the message from buffer*/
       V(Sa);
       }
    }
process C( ) {
     while(true) {
       P(Sc);
       /*receive the message from buffer*/
       V(Sa);
    }
    }
process D( ) {
     while(true) {
       P(Sd);
       /*receive the message from buffer*/
       V(Sa);
    }
 }
coend
```

(2)monitor。

```
TYPE send&receive=monitor
        cond SSb,SSc,SSd,Sb,Sc,Sd; SSb=SSc=SSd=Sb=Sc=Sd=0;
        int SSb_count,SSc_count,SSd_count; SSb_count=SSc_count=SSd_count=0;
        int Sb_count,Sc_count,Sd_count; Sb_count=Sc_count=Sd_count=0;
        bool flagb,flagc,flagd; flagb=flagc=flagd=false;
        message buffer;
InterfaceModule IM;
DEFINE   sendmes,receiveb,receivec,received;
USE   wait,signal,enter,leave;
procedure sendmes( ) {
          enter(IM);
          if(flagb)   wait(Sb,Sb_count,IM);
          if(flagc)   wait(Sc,Sc_count,IM);
          if(flagd)   wait(Sd,Sd_count,IM);
          buffer=message;
          flagb=flagc=flagd=true;
          signal(SSb,SSb_count,IM);
          signal(SSc,SSc_count,IM);
          signal(SSd,SSd_count,IM);
          leave(IM);
    }
procedure   receiveb( ) {
         enter(IM);
       if (flagb==false)   wait(SSb,SSb_count,IM);
           else flagb=false;
           signal(Sb,Sb_count,IM);
         leave(IM);
    }
procedure   receivec( ) {
           enter(IM);
           if (flagc==false)   wait(SSc,SSc_count,IM);
           else flagc=false;
           signal(Sc,Sc_count,IM);
          leave(IM);
    }
procedure   received( ) {
          enter(IM);
           if (flagd==false)   wait(SSd,SSd_count,IM);
           else flagd=false;
           signal(Sd,Sd_count,IM);
          leave(IM);
    }
cobegin
```

```
    process A( ) {
         while(true) {
              produce a message;
              send&receive.sendmes();
             }
         }
    process B( ) {
        while(true) {
              send&receive.receiveb();
             }
         }
    process C( ) {
        while(true) {
              send&receive.receivec();
             }
         }
    process D( ) {
         while(true) {
              send&receive.received();
             }
         }
coend
```

45．试设计一个管程来实现磁盘调度的电梯调度算法。

答：

```
type    diskschedule=monitor
         int headpos;
        enum {up,down} direction;
        bool busy;
        cond S [200];
        int S_count[200];
          headpos=0;direction=up;busy=false;
  InterfaceModule IM;
  DEFINE    request,return;
  USE    wait,signal,enter,leave;
procedure request(int dest;) {
         enter(IM);
         if(busy)    wait(S[dest],S_count[dest],M);
         busy=true;
         if (headpos<dest)  ‖  (headpos==dest&&direction==up)
                direction=up;
            else    direction=down;
         headpos=dest;
         leave(IM);
```

```
    }
procedure return( ) {
    int i;
        enter(IM);
        busy=false;
        if(direction==up)                    /*up为向里方向，即柱面号大的方向*/
                { i=headpos;
                while( i<200 &&S[i]==0)   i++;
                 if(i<200)     signal(S[i],S_count[i],IM);
                   else   {                    /*down为向外方向，即柱面号小的方向*/
                            i=headpos;
                            while(i>=0 &&S[i]==0)   i--;
                              if (i>=0) signal(S[i],S_count[i],IM);
                          }
                }
            else                    /*down为向外方向，即柱面号小的方向*/
              { i=headpos;
                while( i>0 &&S[i]==0)   i--;
                if(i>=0)     signal(S[i],S_count[i],IM);
                  else              /*up为向里方向，即柱面号大的方向*/
                          {i=headpos;
                            while(i<200 &&S[i]==0)   i++;
                            if(i<200 )   signal(S[i],S_count[i],IM);
                       }
                }
        leave(IM);
cobegin
    process visit( ) {
         int k;
        diskschedul．Request(k);
          …
        /*访问第k个柱面*/
          …
        diskschedul．Return( );
          …
    }
coend.
```

46．有 n 个进程将字符逐个读入到一个容量为 80 的缓冲区中（n>1），当缓冲区满后，由输出进程 Q 负责一次性取走这 80 个字符。这种过程循环往复，请用信号量和 P、V 操作写出 n 个读入进程（P1， P2，…Pn）和输出进程 Q 能正确工作的的动作序列。

解：

```
semaphore mutex,empty, full;
```

```
    int count,in;
    char buffer[80];
    mutex=1;empty=80;full=0;
    count=0;in=0;
cobegin
   process Pi( )            //(i=1,...,n)
      while(true){
       /*读入一字符到 x*/
       P(empty);
       P(mutex);
       Buffer[in]=x;
       in=(in+1) % 80;
       count++;
       if (count==80)
          {count=0;V(mutex);V(full);}
       else V(mutex);
       }
     }
process Q( ) {
     M:P(full);
     P(mutex);
     /*读出字符从 buffer[0]*/
     …
     /*读出字符从 buffer[79]*/
     in=0;
     V(mutex);
     for (int j=o;j<80;j++)
          V(empty);
      goto M;
    }
coend.
```

47．设儿童小汽车生产线上有一只大的储存柜，其中有 N 个槽（N 为 5 的倍数且其值≥5），每个槽可存放一个车架或一个车轮。设有三组生产工人，其活动如下：

组 1 工人的活动	组 2 工人的活动	组 3 工人的活动
L1：加工一个车架； 车架放入柜的槽中。	L2：加工一个车轮； 车轮放入柜的槽中。	L3：在槽中取一个车架； 在槽中取四个车轮， 组装为一台小汽车。
goto L1:;	goto L2:;	goto L3:;

试用(1)信号量及 P，V 操作，(2)管程方法正确实现这三组工人的生产合作工作。

解：(1) 信号量及 P，V 操作

将柜子的 n 个槽口分为两部分：N/5 和 4N/5，分别装入车架和车轮

车架 box1[N/5];

```
车轮  box2[4*N/5] ;
semaphore mutex1,mutex2,S1,S2,S3,S4;
int counter,in1,in2,out1,out2;
S1=N/5;S2=4N/5;S3=S4=0;
counter=in1=in2=out1=out2=0;
mutex1=mutex2=1;
cobegin
process worker1( ) {
        while(true) {
            /*加工一个车架*/
            P(S1);
            P(mutex1);
            /*车架放入 box1(in1)*/
            in1=(in1+1) % (N/5);
            V(mutex1);
            V(S3);
            }
      }
process worker2( ) {
            while(true) {
            /*加工一个车轮*/
              P(S2);
              P(mutex2);
              /*车轮放入 box2(in2)*/
              in2=(in2+1) % (4*N/5);
              counter=counter+1;
              if(counter==4)    {counter=0;V(S4);}
              V(mutex2);
              }
      }
process worker3( ) {
            while(true) {
              P(S3);
              P(mutex1);
              /*取车架从 box1(out1)*/
              out1=(out1+1) % (N/5);
              V(mutex1)
              V(S1);
              P(S4);
              P(mutex2);
              /*取车轮从 box2(out2)*/
              out2=(out2+1) % (4*N/5);
              /*取车轮从 box2(out2)*/
              out2=(out2+1) % (4*N/5);
```

```
                /*取车轮从 box2(out2)*/
                out2=(out2+1) % (4*N/5);
                /*取车轮从 box2(out2)*/
                out2=(out2+1) % (4*N/5);
                V(S2);
                V(mutex2);
                /*装配车子*/
                }
          }
coend
```

(2)用管程方法。

```
type    produce_toy_car=monitor
         车架 box1[N/5];
         车轮 box2[4*N/5];
         cond S1,S2,S3,S4;
         int S1_count,S2_count,S3_count,S4_count;
         int counter1,counter2,count,in1,in2,out1,out2;
         counter1=counter2=count=in1=in2=out1=out2=0;
InterfaceModule IM;
DEFINE put1,put2,take;
USE wait,signal,enter,leave;
   procedure put1( ) {
      enter(IM);
      if(counter1==N/5)    wait(S1,S1_count,IM);
      /*车架放入 box1(in1)*/
      in1=(in1+1) % (N/5);
      counter1=counter1+1;
      signal(S3,S3_count,IM);
      leave(IM);
    }
   procedure put2( ) {
         enter(IM);
         if(counter2==(4*N/5))    wait(S2,S2_count,IM);
         /*车轮放入 box2(in2)*/
         in2=(in2+1) % (4*N/5);
         counter2=counter2+1;
         count=count+1;
         if(count==4)    {count=0; signal(S4,S4_count,IM);}
         leave(IM);
      }
    procedure take( ) {
          enter(IM);
         if(counter1==0)    wait(S3,S3_count,IM);
```

```
   /*取车架从 box1(out1)*/
   out1=(out1+1) % (N/5);
   counter1=counter1-1;
   if (counter2<4)    wait(S4,S4_count,IM);
   /*  取车轮从 box2(out2)*/
   out2=(out2+1) % (4*N/5);
   /*取车轮从 box2(out2)*/
   out2=(out2+1) % (4*N/5);
   /*取车轮从 box2(out2)*/
   out2=(out2+1) % (4*N/5);
   /*取车轮从 box2(out2)*/
   out2=(out2+1) % (4*N/5);
   counter2=counter2-4;
   signal(S1,S1_count,IM);
   signal(S2,S2_count,IM);
   leave(IM);
}
```

48 ．某大型银行办理人民币储蓄业务，由 n 个储蓄员负责。每个顾客进入银行后先至取号机取一个号，并且在等待区找到空沙发坐下等着叫号。取号机给出的号码依次递增，并假定有足够多的空沙发容纳顾客。当一个储蓄员空闲下来，就叫下一个号。请用信号量和 P，V 操作正确编写储蓄员进程和顾客进程的程序。

解：

```
    semaphore customer_count,mutex;
    customer_count=0;
    mutex=1;
cobegin
   process customeri( )  {        //(i=1,2,….)
          P(mutex);
          take a number;
          /*等待区找到空沙发坐下*/
          V(mutex);
          V(customer_count);
      }
   Process serversj( ) {         //(j=1,2,3,…)
        while(true) {
         P(customer_count);
         P(mutex);
        /*呼号，被呼号顾客离开沙发走出等待区*/
         V(mutex);
        /*为该号客人服务*/
        /*客人离开*/
       }
```

```
        }
coend
```

49．有一个电子转帐系统共管理 10000 个帐户，为了向客户提供快速转帐业务，有许多并发执行的资金转帐进程，每个进程读取一行输入，其中，含有：贷方帐号、借方帐号、借贷的款项数。然后，把一款项从贷方帐号划转到借方帐号上，这样便完成了一笔转帐交易。写出进程调用 monitor,以及 monitor 控制电子资金转帐系统的程序。

答：

```
TYPE   lock_account=monitor
        bool use[10000];                          /*该帐号是否被锁住使用标志*/
        cond S[10000];   S[10000]=0;               /*条件变量*/
        int S_count[10000]; S_count[10000]=0;
       for (int i=0; i<10000;i++)
           {use[i]=false;}
InterfaceModule IM;
DEFINE lockaccount unlockaccount
USE wait,signal,check,release;
procedure lockaccount(int i,j;) {
          enter(IM)
          if(i>j)
                              { temp= i;
                                i= j;                  /*层次分配,先占号码小的账号
                                j= temp;                  否则可能产生死锁*/
                            }
        if(use [i])     wait(s[i],S_count,IM);
              else    use[i]=true;                     /*锁住 account(i)
        if(use[j])      wait(s[j],S_count,IM);
              else    use[j]:=true;                    /*锁住 account(j)
            leave(IM);
       }
procedure unlockaccount(int i;) {
             enter(IM);
             use [i]=false;
             signal(s[i],S_count,IM);
            leave(IM);
        }
cobegin
       process transferaccount(k) {
                input a information line;
                get the account number i,j and  还款数 x;
                lock-account.lockaccount(i,j)
                 {  按锁住帐号 account(i)和 account(j)执行};
                      A[j]=A[j]-x; A[i]=A[i]+x;
```

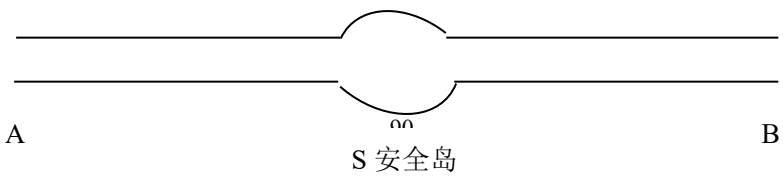
```
                lock-account.unlockaccount(i);
                lock-account.unlockaccount(j);
                }
coend.
```

50. 某寺庙有小和尚和老和尚各若干人，水缸一只，由小和尚提水入缸给老和尚饮用。水缸可容水 10 桶，水取自同一口水井中。水井径窄，每次仅能容一只水桶取水，水桶总数为 3 个。若每次放入、取出水仅为 1 桶，而且不可同时进行。试用一种同步工具写出小和尚和老和尚入水、取水的活动过程。

答：互斥资源有水井和水缸，分别用 mutex1 和 mutex2 来互斥。水桶总数仅 3 只，由信号量 count 控制，信号量 empty 和 full 控制入水和出水量。

```
    semaphore mutex1,mutex2;
    semaphore empty,full;
    int count;
    mutex1=mutex2=1;count=3;empty=10;full=0;
cobegin
    process 打水的小和尚 i( )  {                  //(i=1，2，…)
        while(true) {
         P(empty);              /*水缸满否?*/
         P(count);              /*取得水桶*/
         P(mutex1);             /*互斥从井中取水*/
         /*从井中取水*/
         V(mutex1);
         P(mutex2);             /*互斥使用水缸*/
         /*倒水入缸*/
         V(mutex2);
         V(count);              /*归还水桶*/
         V(full);               /*多了一桶水*/
         }
      }
    process 取水的老和尚 j ( )  {             //(j=1，2，…)
        while(true) {
         P(full);               /*有水吗?*/
         P(count);              /*申请水桶*/
         P(mutex2);             /*互斥取水*/
         /*从缸中取水*/
         V(mutex2);
         V(count);              /*归还水桶*/
         V(empty);              /*水缸中少了一桶水*/
         }
      }
coend.
```

51. 现有一个如图所示的小巷，除安全岛可容 2 人暂时停身外，仅能容 1 人通过，若 A、B 两头都允许行人进和出，试使用信号量与 PV 操作设计一个算法，让两头行人顺利通过小巷。



解：首先，分析此题中的临界资源，显然只能让 A 端巷外的一个行人进入，“入口处”为临界资源，应设信号量 a1 其初值为 1。如果有 2 人及多人进入，那么，只要 B 端有人进入便发生死锁。其次，从 A 至 S 路段，仅能容 1 人通过，“A-S 路段”为临界资源，应设信号量 a2 其初值为 1。同样理由，应设信号量 b1 和信号量 b2，它们的初值均为 1。行人并发执行算法如下。

```
    semaphore a1,a2,b1,b2;
    a1=a2=b1=b2=1;
 cobegin
        process   A( ) {                          process   B() {
            P(a1);                                    P(b1);
            P(a2);                                    P(b2);
            /*行人通过 A-S 段;*/                        /*行人通过 B-S 段*/
            /*到达安全岛*/                              /*到达安全岛*/
            V(a2);                                    V(b2);
            P(b2);                                    P(a2);
            /*行人通过 S-B 段;*/                        /* 行人通过 S-A 段*/
            V(b2);                                    V(a2);
            V(a1);                                    V(b1);
     }                                          }
 coend
```

52 ．在一个分页存储管理系统中，用 free[index]数组记录每个页框状态，共有 n 个页框(index=0，…，n-1)。当 free[index]=true 时，表示第 index 个页框空闲，free[index]=false 时，表示第 index 个页框被占用。试设计一个管程，它有两个过程 acquire 和 return 分别负责分配和回收一个页框。

答：

```
TYPE    framemanagement=monitor
          bool free[n];
          cond waitcondition; waitcondition=0;
          int i,waitcondition_count; waitcondition_count=0;
          for (int index=0; index< n;index++)    free[index]=true;
InterfaceModule IM;
DEFINE    acquire,release;
USE    enter,wait,signal,leave;
procedure acquire(int index) {
              enter(IM);
              for(int i=0;i<n;i++) {
                 if(free[i]) {free[i]=false; index=i;}
                 else    wait(waitcondition,waitcondition_count,IM);
                 }
              leave(IM);
        }
procedure return(int index) {
              enter(IM);
              free[index]=true;
              signal(waitcondition,waitcondition,IM);
              leave(IM);
        }
```

53. AND 型信号量机制是记录型信号量的扩充，在 P、V 操作中增加了与条件“AND”，

故称“同时”P 操作和 V 操作，记为 SP 和 SV(Simultaneous P 和 V)于是 SP(s_1，s_2，...，s_n)和 VS(s_1，s_2，...，s_n)其定义为如下的原语操作：

```
void SP(semaphore  s1，．．．,sn) {
    if(s1>=1 &&  ．．．  && sn>=1)  {
       for(int i=1;i<=n;i++)
           si= si-1;
    }
    else
           /*进程进入第一个遇到的满足 si<1 条件的 si 信号量队列等待，
           同时将该进程的程序计数器地址回退，置为 SP 操作处。*/
    }
void SV(semaphore  s1，．．．,sn) {
    for(int i=1;i<=n;i++)
        si=si+1;
    /*从所有 si 信号量等待队列中移出进程并置入就绪队列。*/
    }
```

试回答 AND 信号量机制的主要特点，适用于什么场合？

答：记录型信号量仅适用于进程之间共享一个临界资源的场合，在更多应用中，一个进程需要先获得两个或多个共享资源后，才能执行其任务。AND 型信号量的基本思想是：把进程在整个运行其间所要的临界资源，一次性全部分配给进程，待该进程使用完临界资源后再全部释放。只要有一个资源未能分配给该进程，其他可以分配的资源，也不分配给他。亦即要么全部分配，要么一个也不分配，这样做可以消除由于部分分配而导致的进程死锁。

54. 试用 AND 型信号量和 SP、SV 操作解决生产者-消费者问题。

答：

```
item B[k];
semaphore sput;      /*指示有可用的空缓冲区的信号量 */
semaphore sget;      /*指示缓冲区有可用的产品信号量 */
semaphore mutex;     /* 互斥信号量*/
mutex=1;
sput= k;             /* 缓冲区允许放入的产品数*/
sget= 0;             /* 缓冲区内没有产品 */
int in= 0;
int out= 0;
cobegin
   process producer_i( ) {
      while(true) {
      produce a product;
       SP(sput,mutex);
       B[in]=product;
       in=(in+1) % k;
       SV(mutex,sget);
     }
}
process consumer_j ( ) {
     while(true) {
     SP(sget,mutex);
```

```
            product= B[out];
            out=(out+1) % k;
            SV(mutex,sput);
            consume a product;
           }
        }
      Coend
```

55. 试用 AND 型信号量和 SP、SV 操作解决五个哲学家就餐问题。

答：

```
    semaphore fork_i[0..4];
    fork_i=1;
cobegin
process philosopheri( ) {            /*i=0,1,2,3*/
    L1:
    /*思考*/
    SP(fork[i],fork[(i+1)%5]);       /*i=4 时，SP(fork[4],fork[0])*/
    /*吃通心*/;
    SV(fork[i],fork[(i+1)%5]);
    goto L1;
 }
coend
```

56. 如果 AND 型信号量 SP 中，并不把等待进程的程序计数器地址回退，亦即保持不变，则应该对 AND 型信号量 SV 操作做何种修改?

答：要保证进程被释放获得控制权后，能再次检测每种资源是否>=1。故可在 else 部分增加一条 go to 语句，转向 if 语句再次检测每种资源状况。

57. 通用型信号量机制是对 AND 型信号量机制作进一步扩充形成的，SP($s_1,t_1,d_1;...;s_n,t_n,d_n$)和 SV($s_1,d_1;...s_n,d_n$)的定义如下：

```
   void SP(s1,t1,d1;...;sn,tn,dn) {
     semaphore s1,...,sn;
    int  t1, ...,tn;
    int  d1, ...,dn;
    if(s1>=t1&& ...&&sn>=tn)
          for(int i=1;i<=n;i++)
             si=si-di;
     else
         /*进程进入第一个遇到的满足 si<ti 条件的 si 信号量队列等待，
         同时将该进程的程序计数器地址回退，置为 SP 操作处。*/
     }
   void SV(s1,d1;...;sn,dn) {
       semaphore s1,...sn;
        int d1 ,...,dn;
```

```
        for(int i=1;i<=n;i++)
          si=si+di;
           /*从 si 信号量等待队列中移出所有进程并置入就绪队列。*/
      }
```

其中，t_i 为这类临界资源的阀值，d_i 为这类临界资源的本次请求数。

试回答通用型信号量机制的主要特点，适用于什么场合？

答：在记录型和同时型信号量机制中，P、V 或 SP、SV 仅仅能对信号量施行增 1 或减 1 操作，每次只能获得或释放一个临界资源。当一请求 n 个资源时，便需要 n 次信号量操作，这样做效率很低。此外，在有些情况下，当资源数量小于一个下限时，便不预分配。为此，可以在分配之前，测试某资源的数量是否大于阀值 t。对 AND 型信号量机制作扩充，便形成了通用型信号量机制。

58. 下面是通用型信号量的一些特殊情况：

(1)SP(s,d,d)

(2)SP(s,1,1)

(3)SP(s,1,0)

试解释它们的物理含义和所起的作用。

答：

(1)SP(s,d,d)　此时在信号量集合中只有一个信号量、即仅处理一种临界资源，但允许每次可以申请 d 个，当资源数少于 d 个时，不予分配。

(2)SP(s,1,1)　此时信号量集合已蜕化为记录型信号量(当 s>1 时)或互斥信号量(s=1 时)。

(3)SP(s,1,0)　这是一个特殊且很有用的信号量，当 s>=1 时，允许多个进程进入指定区域；当 s 变成 0 后，将阻止任何进程进入该区域。也就是说，它成了一个可控开关。

59. 试利用通用型信号量机制解决读者-写者问题。

答：对读者-写者问题作一条限制，最多只允许 rn 个读者同时读。为此，又引入了一个信号量 L，赋予其初值为 rn，通过执行 SP(L,1,1)操作来控制读者的数目，每当一个读者进入时，都要做一次 SP(L,1,1)操作，使 L 的值减 1。当有 rn 个读者进入读后，L 便减为 0，而第 rn+1 个读者必然会因执行 SP(L,1,1)操作失败而被封锁。

利用通用型信号量机制解决读者-写者问题的算法描述如下：

```
var    int rn;                    /*允许同时读的读进程数
       semaphore L=rn;            /*控制读进程数信号量,最多 rn
       semaphore W=1;
  cobegin
          process reader( ) {
             while(true) {
                 SP(L,1,1 ;W,1,0);
                 {Read the file};
                 SV(L,1);
              }
          process writer( ) {
             while(true) {
                SP(W,1,1;L,rn,0);
                {Write the file};
                SV(W,1);
```

```
        }
      }
coend
```

上述算法中，SP(W,1,0) 语句起开关作用，只要没有写者进程进入写，由于这时 W=1，读者进程就都可以进入读文件。但一旦有写者进程进入写时，其 W=0，则任何读者进程及其他写者进程就无法进入读写。SP(W,1,1;L,rn,o) 语句表示仅当既无写者进程在写(这时 W=1) 、又无读者进程在读(这时 L=rn) 时，写者进程才能进行临界区写文件。

60. 试利用二元信号量机制解决生产者-消费者问题。
解：

```
item Buffer[k];
binary_semaphore BS =1;  /*互斥信号量 */
binary_semaphore BD=0;   /*等待信号量 */
int count=0;
int in=0;
int out=0;
void producer( ) {
    While(true) {
          produce( );
          BP(BS);
          Append( ) to Buffer[in];
          in=(in+1) % k;
          count++;
          if(count==1) BV(BD);
          SV(BS);
       }
   }
 void consumer( ){
    BP(BD);
    While(true) {
        int i;
       BP(BS);
        Take( ) from Buffer[out];
       out=(out+1) % k;
        i=count--;
        BV(BS);
        consume( );
        if(i==0) BP(BD);
      }
  }
  cobegin
  Procucer( );Consumer( );
  Coend
```

61 . 试利用二元信号量机制来实现一般信号量。
解：
P（s）和 V(s)定义：

```
binary_semaphore mutex=1;
binary_semaphore delay=0;
int s;

P(s) {
   BP(mutex);
   s--;
   if(s<0) {
         BV(mutex);
         BP(delay);
         }
   BV(mutex);
}

V(s) {
 BP(mutex);
 s++;
 if(s<=0)
       BV(delay);
 else
       BV(mutex);
 }
```

62. 试利用关中断实现单处理器上的信号量机制。

解：

```
typedef struct {
    int value;                  /*信号量值*/
    struct pcb *list;           /*信号量队列指针*/
 }semaphore;
 void P(s) {
      inhibit interrupts;
      s.value--;
      if(s.value<0) {
         W(s.list);allow interrupts;
    }
    else
       allow interrupts;
}
 void V(s) {
     inhibit interrupts;
     s.value++;
     if(s.value<=0) R(s.list);
     allow interrupts;
 }
```

63．试利用 test&set 指令实现单处理器上的信号量机制。

解：

```
typedef struct {
    int flag;                   /*标志*/
```

```
    int value;                  /*信号量值*/
    struct pcb *list;           /*信号量队列指针*/
  }semaphore;
  void P(s) {
      while(!test_set(s.flag))
      s.value--;
      if(s.value<0) {
         W(s.list); s.flag =0;
      }
        s.flag=0;
}
 void V(s) {
      while(!test_set(!s.flag))
      s.value++;
      if(s.value<=0) R(s.list);
      s.flag=0;
 }
```

64.试用管程实现睡眠的理发师问题。

解：

```
type   sleepy_barber=monitor
         cond stool; stool=0            /*理发椅，无顾客时，理发师睡眠*/
         cond chair; chair=0;           /*顾客多时，让等候顾客座的椅子，chair<N*/
         int stool_count; stool_count=0;
         int chair_count; chair_count=0;
         int waiting=0;                 /*等候顾客个数*/
InterfaceModule IM;
DEFINE sleepy,customer_enter,customer_leave;
USE enter,wait,signal,leave;
  procedure sleep( ) {
     enter(IM);
     if(waiting==0)   wait(stool,stool_count,IM);
     leave(IM);
     }

  procedure customer_enter( ) {
       enter(IM);
       if(waiting<N)   {
          waiting++;
          if(waiting==1)   signal(stool,stool_count,IM);
          else       wait(chair,chair_count,IM);
          }
       else
            /*人满了，顾客离开*/
```

```
        leave(IM);
    }
  procedure customer_leave ( ) {
        enter(IM);
        waiting--;
         if(waiting<>0)    signal(chair,chair_count,IM)
         }
        leave(IM);
    }

    cobegin
    process barber( ) {
        while(true) {
           sleepy_barber.sleep( );
           cut    hair（）;
     }；
    process customer（）    {
         while（true）    {
                sleepy_barber. customer_enter( ) ；
                get_haircut（）；
                sleepy_barber. customer_leave( ) ；
                }
    }
    coend
```

65. Bakery 算法(Lamport,1974)是解决 n 个进程访问临界区问题的一种方法。

```
 bool choosing[n];
    int number[n];
  while(true) {
    choosing[i]=true;
    number[i]=1+max(number[0],number[1],…,number[n-1]);
    choosing[i]=false;
      for(int j=0;j<n;j++)    {
            while(choosing[j]);
            while(number[j]!=0&&(number[j],j)<(number[i],i));
          }
      <critical section>;
       number[i]=0;
      <remainder>;
  }
```

今定义：

(1) (a,b)<(c,d)定义为 a<c ‖ (a=c && b<d)。

(2)choosing[i]和 number[i]可以被 process i 读和写，但只可被 process[j](j!=i)读。

(3)数组 choosing 初始化为 false，而数组 number 初始化为 0。

请用自然语言描述 Bakery 算法，并说明：

(1)Bakery 算法是如何解决互斥访问临界区问题的?

(2)Bakery 算法是如何解决死锁的?

解：

(1)Bakery 算法类似于面包店对顾客的一种服务行为。每个顾客进入面包店时都会领到一个号码，服务员为拥有最小号码的顾客服务。

先对算法作一个分析。为叙述方便对程序进行编号，

```
choosing[i]=true;                    ……                    .①
number[i]=1+max(number[0],number[1],…,number[n-1])……      .②
choosing[i]=false;…… …… … .③
  for(int j=0;j<=n;j++)   {
        while(choosing[j]); ……   …… . .④
        while(number[j]!=0 && (number[j],j)<(number[i],i));    … ⑤
    }
   <critical section>;…… .⑥
   number[i]=0;……    ..⑦
```

进程用 Bakery 算法进入临界区经过两个阶段：

步 1 进程进入“面包店入口”，选择一个号码。具体做法分两种情况：

(1) 进程先后进入“面包店入口”， 见①，它们先读取其他进程的号码，并选择一个比其他进程 number 的最大值加 1 的数字作为自己的 number 号，见②。

(2) 进程并发进入“面包店入口”，这时因交叉执行，可能有若干个进程读取到相同的 max 值，于是加 1 后，可能出现多个进程，如 process i 和 process j，它们的 number[i]=number[j]=max(number[0],number[1],…,number[n-1])+1。

步 2 process i 采用以下方法检查可否进入临界区。

(1) 对任意其它进程 pj，Pi 首先检查 pj 是否已经在“面包店入口”，如果在(即①的位置)，那么，这时 choosing[j]=true，故 pi 执行④，等待 pj 退出“面包店入口”。

(2) 接着 pi 执行⑤，等待条件 number[j]!= 0 and (number[j],j)<(number[i],i) 不满足。由于(a,b)<(c,d) 定义为 a<c ‖ (a=c && b<d)。故分两种情况讨论：

1) number[j]!=0 && number[j]<number[i] 也就是当 number[j]比 number[i]小时，pi 应等待直到 process j 执行结束退出临界区，process j 执行结束后 number[j]便=0。

2) number[j]!= 0 and number[j]=number[i] && j<i 也就是当并发进程的 number 值相等时，只要 j<i，那么，pi 应等待直到 process j 执行结束退出临界区，这时是按进程号大小排序，小的先做。

步 3 当 pi 对所有其他进程验证了上述条件，便可进入临界区。

然后，证明两个结论。

结论 1 若进程 pi 已在临界区，且若干其他进程已选择了号码，那么，(number[i],i)<(number[k],k)成立。

证明 若 pi 在临界区，则一定经历了 k 轮 for 循环，而且，要么 number[k]=0，要么(number[i],i)<(number[k],k)。首先，假设 pi 读出的 number[k]的值=0，也就是说 pk 还未选号。

1) 此时若 pk 不在“面包店入口”，那么它可读到最新的 number[i]值，故确保 number[k]>number[i]。

2) 此时若 pk 已在“面包店入口”，那么说明这次进入一定是在 pi 检查了 choosing[k]

之后，因为，pi 在检查条件(number[k]=0 && (number[i],i)<(number[k],k)之前需等待 pk 完成选择号码。这也意味着 pk 将读到最新的 number[i]，因此，number[i]<number[k]。

如果在 k 轮循环中，(number[i],i)<(number[k],k)，那么这个结论将保持，因为 number[i]的值不会改变，而 number[k]的值只会增加。

结论 2 若进程在临界区中，则 number[i]>0。这个结论是显然的。

(2)Bakery 算法能解决互斥访问临界区问题。

如果两个进程 pi 和 pk 都进入了临界区，则他们的 number 均>0，则由结论 1 得到(number[i],i)<(number[k],k)。反之，有(number[i],i)>(number[k],k)。矛盾。

(3)Bakery 算法能解决死锁问题

任一时都有进程(number[i],i)，或者它的 number[i]值最小(但>0)，或者 number[i]与其他一些进程相等，但其进程号 i 最小，故 pi 能在有限时间内执行结束。以此类推，系统不会发生死锁。

66. 荷兰数学家 T.Dekker 提出的 Dekker 算法如下：

```
var  bool inside[2];
    int turn ;
    turn = 0 or 1;
    inside[0]=false;
    inside[1]=false;
main( ) {
cobegin
process P1( ) {
    inside[0]=true;
    while (inside[1]) {
                  if (turn==1) {
                  inside[0]=false;
                  while(turn==1);
                  inside[0]=true;
                 }
                }
    {临界区};
    turn = 1;
    inside[0]=false;
}
process P2( ) {
     inside[1]=true;
     while( inside[0]) {
                    if (turn==1) {
                    inside[1]=false;
                    while(turn==0);
                    inside[1]=true;
                   }
                   }
    {临界区};
    turn = 0;
    inside[1]=false;
 }
coend
```

试解释该算法的执行过程，及说明能保证进程互斥地进入临界区。

解：荷兰数学家 T.Dekker 算法能保证进程互斥地进入临界区，这是最早提出的一个软件互斥方法，此方法用一个指示器 turn 来指示应该哪一个进程进入临界区。若 turn=1 则进程 P1 可以进入临界区；若 turn=2 则进程 P2 可以进入临界区。

Dekker 算法的执行过程描述如下：当进程 P1(或 P2)想进入自己的临界区时，它把自己的标志位 inside1（或 inside2）置为 true，然后，继续执行并检查对方的标志位。如果为 false，表明对方不在也不想进入临界区，进程 P1（或 P2）可以立即进入自己的临界区；否则，咨询指示器 turn，若 turn 为 1（或为 2），那么，P1（或 P2）知道应该自己进入，反复地去测试 P2（或 P1）的标志值 inside2(或 inside1)；进程 P2（或 P1）注意到应该礼让对方，故把其自己的标志位置为 false，允许进程 P1（或 P2）进入临界区；在进程 P1（或 P2）结束其临界区工作后，把自己的标志置为 false，且把 turn 置为 2（或 1），从而，把进入临界区的权力交给进程 P2（或 P1）。

这种方法显然能保证互斥进入临界区的要求，这是因为仅当 turn= i(i = 1，2)时进程 Pi (i = l，2)才能有权力进入其临界区。因此，一次只有一个进程能进入临界区，

且在一个进程退出临界区之前，turn 的值是不会改变的，保证不会有另一个进程进入相关临界区。同时，turn 的值不是 1 就是 2，故不可能同时出现两个进程均在 while 语句中等待而进不了临界区。

67 .Eisenberg and McGuire 的 N 个并发进程的临界区互斥算法如下:

```
INITIALIZATION:
        shared enum states {IDLE,WAITING,ACTIVE} flags[n-1];
        shared int turn;
        int index;
        turn=0;
        for (index=0;index<n;index++) {
                flags[index]=IDLE;
        }
ENTRY PROTOCOL(for Process i):
        do {
                flags[i]=WAITING;
                index=turn;
                while(index!=i) {
                        if (flag[index]!=IDLE) index=turn;
                        else index=（index+1）%n;
                }
                flags[i]=ACTIVE;
                index=0;
                while((index<n)&&((index==i)||(flags[index]!=ACTIVE))) {
                        index = index+1;
                }
          } while(!((index>=n)&&((turn==i)||(flags[turn]==IDLE))));
        turn=i;
EXIT PROTOCOL(for Process i ):
        index=(turn+1)%n;
        while(flags[index]==IDLE) {
                index=（index+1）%n;
        }
        turn=index;
        flag[i]=IDLE;
```

(1)试说明 n 个进程能正确实现临界区互斥算法的设计思想；(2)对每个语句或程序段进行注释。

解：(1)临界区互斥算法的设计思想

今有进程 P_0 至 P_{n-1}，每个可能有三种状态：“空状态”、“等待态”和“活跃态”。turn 是访问令牌，只有当 turn 设为进程号时，该进程才有可能进入 CS，而使其余进程在申请获得 turn 时陷入忙式等待。

• 互斥进入—互斥由两个循环测试实现：

P_i 想进入 CS 时，将状态置为“等待态”，反复扫描从 P_{turn} 至 P_i 之间的所有进程的状态是否为“空状态”？可能有多个排在其左边的其他进程（P_{turn}-P_{i-1}）与 P_i 在竞争，当确信所有这些进程均为“空状态”后，进程 P_i 才把自己设为“活跃态”。

然而，仍然有其他进程，虽然扫描开始时间迟，但却抢先进入“活跃态”，对于 P_i 来说只好反复测试和检查，直到所有其他进程(实质为排在其右边的其他进程)均不为“活跃态”，否则从第一个循环开始再重新测试。当确信自己是最优先的一个进程时才能进入 CS。

• 无空等待—等待进程反复执行忙式测试，且从左向右次序循环排序，当 P_{turn}-P_{i-1} 进程执行结束后便能轮到 P_i 进入 CS，故为有限等待。

• 有空让进—进程退出 CS 时，如果有非空状态进程，便将释放右边的第一个陷于忙式等待的进程，将访问令牌交给它，因为令牌按序传递故不会发生饥饿。

（2）注释：

INITIALIZATION:“

/* turn 是访问令牌，其值为 0 到 n-1 之间，指出某进程可以访问 CS。index 为工作变量，值在 0 到 n 之间。初始化程序让所有进程处于“空状态”。turn 的初值为 0，实际上可初始化为 0 至 n-1 间的任一个数。 */

```
shared enum states {IDLE,WAITING,ACTIVE} flags[n-1];
shared int turn;
int index;
turn=0;
        for(index=0;index<n;index++) {
        flags[index]=IDLE;
}
```

ENTRY PROTOCOL(for Process *i*):

```
do {
```

/*进入 CS 规则-进程 P_i 发布欲进入 CS，先把状态改为“等待态”。*/

```
flags[i]=WAITING;
```

/*从 turn 指向的进程 P_{turn} 开始，直到 P_i 本身为止，扫描所有进程的状态，直到它们均为“空状态”。只要其中有一个进程为非空状态，则反复地执行循环 1。如果所查进程非“空状态”，说明它已进入 CS 或想进入 CS，那么，把 turn 赋给 index，从 turn 开始再次 while 循环，等待某进程退出临界区。否则，继续循环直查到 P_{i-1}。turn 的初值可<i，turn 初值也可能>i，这时，检查要从右边绕到左边，直至 i-1。 */

```
index=turn;
while(index!=i) {
        if(flag[index]!=IDLE) index=turn;
        else index=（index+1）%n;
}
```

/*现在，从 turn 至 i-1 的进程均为“空状态”，可把 P_i 的状态改为”活跃态”，若仅有它为”活跃态”才能进入 CS。 */

```
flags[i]=ACTIVE;
```

/*因为并发进程在执行，判断是否有其他进程为“活跃态”。 */

```
index=0;
1.      while((index<n)&&((index==i)||(flags[index]!=ACTIVE))){
                index=index+1;
        }
    }while(!((index>=n)&&((turn==i)||(flags[turn]==IDLE))));
```

/*若有其他进程为“活跃态”，则从第一个循环起从头再查，否则，把 i 赋给 turn，可轮到 Pi 进 CS。 */

```
Turn=i;
```

EXIT PROTOCOL(for Process *i*):

```
 /*从 i+1 开始循环找一个非“空状态”进程，让它进入 CS。若没有这样的进程，
则 turn 保持为本次的值 i。                                    */
        index=(turn+1)%n;
        while(flags[index]==IDLE) {
                index=(index+1)%n;
        }
/*turn 指向右边需要进入 CS 的第一个进程或 i。                  */
        turn=index;
/*Pi 状态改为“空状态”，即 Pi 退出临界区。                      */
        flag[i]=IDLE;
```