

# CH3 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 P、V 操作写出三个进程正确工作的流程。

答：

```
1 ) var B : array [ 0 , k-1 ] of item ;
    sread : semaphore := k ;
    smanage : semaphore := 0 ;
    swrite : semaphore := 0 ;
    rptr : integer := 0 ;
    mptr : integer := 0 ;
    wptr : integer := 0 ;
    x : item

cobegin
process reader ;           process manager ;           process
writer ;
begin                       begin                       begin
L1 : read a message into x ;   L2 : P ( smanage ) ;           L3 : P
( swrite ) ;
P ( sread ) ;                               x:=B[mptr];
x:=B[swrite];
B[rptr]:=x;                               mptr:=(mptr+1) mod k;
wptr:=(wptr+1) mod k;
Rptr:=(rptr+1) mod k;                     manage the message in x;
V(sread);
V(smanage);                               B[mptr]:=x;
print the message in x;
Goto L1;                                   V(swrite);
goto L3;
End;                                       goto L2;
end;
```

End;

```
coend
2 ) var A , B :array [ 0 , k -1 ] of item ;
sPut1 : semaphore:=k;
SPut2: semaphore:=k;
sget1 : semaphore := 0 ;
sget2 : semaphore := 0 ;
put1 : integer :=0 ;
put2 : integer := 0 ;
get1 : integer :=0 ;
get2 : integer := 0 ;
cobegin
```

```

process reader ;
process Writer ;
begin
begin
L1 : read a message into x ;      L2      : P ( sget1 ) ;      L3      :
P ( sget2 ) ;
P ( SPut1 ) ;      x      : = A [ get1] ;      x      :
= B [get2];
A [put1]:=x ;      get1 : (get1+1 ) mod k ;
get2:= ( get2 + 1 ) mod k ;
Put1:=(put1+1) mod k ;      V(sput1);
V(sput2);
V(sget1);      manage the message into x;
print the message in x;
Goto L1;      P(sput2);
goto L3;

      Put2:=(put2+1) mod k;
      V(sget2);
      Goto L2;
      End;
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 ( $m \leq n$ ) 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:

Begin

begin

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;
End	end

答：现对进程语句进行编号，以方便描述。

P1 :	P2 :
begin	begin
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 ;
End	end

、 和 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 时，可以得到  $x = 10$ ， $y = 4$ 。按 Bernstein 条件，语句 的执行结果不受语句 的影响，故语句 执行后得到  $z = 5$ 。最后，语句 和 并发执行，这时得到了两种结果为：

语句 先执行： $x = 10, y = 9, z = 150$

语句 先执行： $x = 10, y = 19, z = 15$

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

7 :  $z = X$ ;

$z := y + 1$ ;

$y := Z + y$ ;

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 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 P、v 操作：

```
var name : array [1, 100] of A ;  
A = record  
number : integer ;  
name: string ;  
end  
for i := 1 to 100 do {A [ i ].number : i ; A [ i ].name : null;}
```

```
mutex , seatcount : semaphore ;  
i : integer ; mutex := 1 ; seatcount := 100 ;  
cobegin  
{  
process readeri ( var readename : string ) ( i=1 , 2 , )  
{  
P ( seatcount ) ;  
P ( mutex ) ;  
for i := 1 to 100 do i++
```

```
if A [ i ].name = null then A [ i ].name : readename ;
```

```
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 ) 使用管程操作：

TYPE readbook=monitor

```

VAR R: condition ;
l,seatcount      : integer;
name: array [ 1:100] of string ;
DEFINE rcadercome, readerleave ;
USE check , wait , signal , release ;
Procedure readercome ( readername )
begin
check ( IM ) ;
if seatcount      100 wait ( R,IM )
seatcount := seatcount + 1 ;
for i=1 to 100 do i++
if name[i] ==null then name[i]:= readername;

get the seat number = i ;
release ( IM ) ;
end
procedure readerleave ( readername )
begin
check ( IM ) ;
seatcount--;
for i = 1 to 1 00 do i++

if name  [ i  ] readername then name  [ i  ] :null;

release ( IM ) ;
end
begin
seatcount := 100 ; name:      = null ;
end
cobegin
{
process readeri ( i = 1 , 2      . ,    )
begin
readercome ( readername      ) ;
read the book ;
readerleave ( readername      ) ;
leave the readroom;

end
}

coend.

```

5. 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。

```
var S1, S2 : semaphore;
```

```
S1 := 1; S2 := 0;
```

```
cobegin
```

```
{
```

```
    process P1
```

```
    begin
```

```
        repeat
```

```
            P( S1 );
```

```
            拣白子
```

```
        V ( S2 );
```

```
    until false ;
```

```
end
```

```
process P2
```

```
begin
```

```
repeat
```

```
P ( S2 );
```

```
拣黑子
```

```
V ( S1 );
```

```
until false ;
```

```
end
```

```
}
```

```
coend .
```

答 2：

```
TYPE pickup-chess = MONITOR
```

```
VAR flag : boolean ;
```

```
S-black , s-white : condition ;
```

```
DEFINE pickup-black , pickup-white ;
```

```
USE wait,signal , check , release ;
```

```
procedure pickup-black ;
```

```
begin
```

```
check(IM) ;
```

```
if flag then wait(s-black,IM) ;
```

```
flag : = true;
```

```

pickup a black;

signal(S-white,IM);

release ( IM ) ;
end
procedure pickup-white ;
begin
check ( IM ) ;
if not flag then wait(S-white,IM );

flag :=false ;
pickup a white ;
signal ( S-black,IM ) ;
release ( IM ) ;
end

begin
flag:=true ;
end

main ( )
{ cobegin
    process -B ( ) ;
    process -W ( ) ;
    coend
}
process-B ( )
begin
pickup-chess.pickup-black ( ) ;

other ;
end
process-W ( )
begin
pickup-chess.pickup-white( ) ;

other ;
end

```

6 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。

7 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：

```
var S1, S2 : semaphore ;
```

```
S1=0; S2=0;
```

```
cobegin
```

```
{
```

```
driver ( ) ;
```

```
busman ( ) ;
```

```
}
```

```
coend
```

```
driver ( )
```

```
begin
```

```
while ( 1 ) {
```

```
P ( S1 )
```

```
    启动车辆；正常行车；到站停车；
```

```
V ( S2 ) ;
```

```
}
```

```
end
```

```
busman ( )
```

```
begin
```

```
while ( 1 ) {
```

```
    关车门；
```

```
V ( S1 )
```

```
    售票；
```



P ( S2 )

开车门 ;

上下乘客 ;

}

end

8、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 ) 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。

var S1 , S2 ,S3 , S4 : semaphore ;

S1 := 1 ;S2 :=S3 := S4 := 0 ;

cobegin

{ process P1

begin

repeat

有顾客到来 ;

P ( S1 ) ;

接受顾客点菜 ;

V ( S2 ) ;

until false ;

end

process P2

begin

repeat

P ( S2 ) ;

准备顾客的饭菜 ;

v ( S3 ) ;

until false ;

end

process P3

begin

repeat

```
P (S3 ) ;  
将做好的饭菜打包 ;
```

```
V ( S4 ) ;  
until false ;
```

```
end
```

```
process P4  
begin  
repeat  
P( 54 ) ;  
收款并提交食品 ; V ( 51 ) ;  
until false ;  
end  
}  
coend .
```

9、在信号量  $S$  上作  $P$ 、 $v$  操作时， $S$  的值发生变化，当  $S > 0$ 、 $S=0$ 、 $S < 0$  时，它们的物理意义是什么？

答： $S$  的值表示它代表的物理资源的使用状态： $S > 0$  表示还有共享资源可供使用。 $S = 0$  表示共享资源正被进程使用但没有进程等待使用资源。 $S < 0$  表示资源已被分配完，还有进程等待使用资源。

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
begin	begin
A ;	D ;
B ;	E ;
C ;	end ;
end ;	

( 2 ) 两个并发进程  $P1$  和  $P2$  并发执行，它们的程序分别如下：

$P1$	$P2$
repeat	repeat
$k:=k \times 2$ ;	print $k$ ;
$k:=k+1$ ;	$k:=0$ ;
until false ;	until false ;

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：

P1	P2
repeat	repeat
k:=k × 2 ;	printk ;
k:=k+1 ;	k:=0 ;
until false ;	until false ;

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：

、	、	、	，这时的打印值为：	47
、	、	、	，这时的打印值为：	23
、	、	、	，这时的打印值为：	46
、	、	、	，这时的打印值为：	46
、	、	、	，这时的打印值为：	23
、	、	、	，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：

(1) 用信号量实现管程；

(2) 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：

```
Var mutex,c:semaphore ;
mutex:=1 ; c:=0 ;
void enter-monitor ( ) /*      进入管程代码，保证互斥

P ( mutex ) ;
}
void leave-monitor-normally ( )/*      不发信号退出管程
{
V ( mutex ) ;
}
```

void leave-with-sigal(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 ) ;  
}

void 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=monitor

VAR S ; condition ;

C:integer ;

DEFINE P , V ;

USE check , wait , signal , release ;

procedure P

begin

check ( IM ) ;

C:= C-1 ;

if C < 0 then wait ( S,IM ) ;

release ( IM ) ;

end

procedure V

begin

check ( IM ) ;

C := C + 1 ;

if C >= 0 then signal ( S,IM ) ;

release ( IM ) ;

end

begin

C:=初值；

End.

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, 就可以用消息传递实现管程。

( 2 ) 用管程实现消息传递。

TYPE mailbox=monitor

VAR r , k , count:integer ;

buffer : array[0 , n-1] of message ;

full , empty:condition ;

DEFINE add , get ;

USE check , wait , signal , release ;

procedure add ( r ) ;

begin

check ( IM ) ;

if count=n then wait ( full,IM ) ;

buffer [r]:=message ;

r: = (r+1) mod n

count:=count + 1 ;

if count = 1 then signal ( empty , IM ) ;

release ( IM ) ;

end

procedure get ( m ) ;

begin

check ( IM ) ;

if count = 0 then wait ( empty , IM ) ;

m:=buffer [ k ] ;

count : = count-1 ;

if count = n-1 then signal ( full , IM ) ;

release ( IM ) ;

end

begin

r:= 0 ; k:= 0 ; count:=0 ;

end

13 证明信号量与消息传递是等价的 :

( 1 ) 用信号量实现消息传递 ;

( 2 ) 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 使用 ( 1 ) 消息传递， ( 2 ) 管程， 实现生产者和消费者问题。答： ( 1 ) 见课文 ch3 3.5.4 节。( 2 ) 见课文 Ch3 3.4.3 节。

15 试利用记录型信号量和 P、V 操作写出一个不会出现死锁的五个哲学家进餐问题的算法。答：

```
var fork:array [0 , 4] of semaphore ;
fork:=1 ;
cobegin
{
  process Pi /* i = 0 , 1 , 2 , 3 */
begin
L1 :
思考 :
P(fork[i]) ; /* i =4,P(fork [0]) */
P(fork[(i+1) mod 5]) /* i =4P ( fork [4] ) */
吃通心面 ;
V (fork[i] ;
V (fork[(i+1) mod 5] ) ;
goto L1 ;
end ;
}
coend ;
```

16 Dijkstra 临界区软件算法描述如下：

```
var flag : array[0 , n] of (idle,want-in , in_cs ) ;
```

```

turn:integer ; tune:0 or 1 or      , or , n-1 ;
process Pi(i=0,1      , , n-1)
var j ; integer ;
begin
  repeat
    repeat
      flag [i] :want_in ;
      while turn      1 do

          if flag[turn]==idle then turn:=i ;
          flag[i]:= ip_cs ;
          j:=0 ;
          while (j < n ) & (j==1 or flag[j]      in_cs )
          do j:=j + 1 ;
          until j      n :
          critical section ;
          flag [i]:=idle ;

          ,
          until false ;

end .

```

试说明该算法满足临界区原则。

答：为方便描述，把 Dijkstra 程序的语句进行编号：

```

repeat
  flag[i]:=want_in      ;
  while turn      i do
    if flag[trun]==idle then turn:=i      ;
    flag[i]: = in_cs      ;
    j:= 0 ;
    while(j < n ) & (j==1 or flag[j]      in_cs )
    do j:=j + 1 ; @
    until j      n ;
    critical section ;
    flag[i] :=idle      ;

,

```

(1 ) 满足互斥条件

当所有的巧都不在临界区中，满足  $flag[j] \neq in\_cs$ （对于所有  $j, j \neq i$ ）条件时， $P_i$  才能进入它的临界区，而且进程  $P_i$  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  $P_i$  总是先置自己的  $flag[i]$  为  $in\_cs$  后，才去判别  $P_j$  进程的  $flag[j]$  的值是否等于  $in\_cs$  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  $P_i$  在执行进入临界区代码时先执行语句  $flag[i] = in\_cs$ ，其相应的  $flag[i]$  的值不会是  $idle$ 。注意到  $flag[i] = in\_cs$  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  $P_0$  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  $P_j (j=1, 2, \dots, n-1)$  交错执行语句  $flag[j] = want\_in$  后（这时  $flag[j] = want\_in$ ），再做语句  $turn = j$ （第一个  $while$  语句），来查询  $flag[turn]$  的状态。显然，都满足  $turn = i$ ，所以，都可以执行语句  $flag[i] = in\_cs$ ，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  $turn = k (1 \leq k \leq n-1)$ 。接着，进程  $P_j (j=1, 2, \dots, n-1)$  交错执行语句  $flag[j] = want\_in$ ，于是最多同时可能有  $n-1$  个进程处于  $in\_cs$  状态，但不要忘了仅有一个进程能成功执行语句  $flag[i] = in\_cs$ ，将  $m$  置为自己的值。

假设  $\{P_1, P_2, \dots, P_m\}$  是一个已将  $flag[i]$  置为  $in\_cs (i=1, 2, \dots, m)$  ( $m \leq n-1$ ) 的进程集合，并且已经假设当前  $turn = k (1 \leq k \leq m)$ ，则  $P_k$  必将在有限时间内首先进入临界区。因为集合中除了  $P_k$  之外的所有其他进程终将从它们执行的语句  $flag[j] = want\_in$ （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  $flag[i] = want\_in$  重新执行，再次置  $flag[i] = want\_in$ ，继续第二轮循环，这时的情况不同了， $flag[turn] = flag[k]$  必定  $idle$ （而为  $in\_cs$ ）。而进程  $P_k$  发现最终除自身外的所有进程  $P_j$  的  $flag[j]$   $in\_cs$ ，并据此可进入其临界区。

17 另一个经典同步问题：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。

```
vars, S1, S2, S3; semaphore;
```

```
S:=1; S1:=S2:=S3:=0;
```

```
fiag1, flag2, fiag3: Boolean;
```

```
fiag1:=flag2:=flag3:=true;
```

```
cobegin
```

```
{
```

```
    process  供应者
```

```
    begin
```

```
        repeat
```

```
            P(S);
```

```
            取两样香烟原料放桌上，由 flagi 标记； / * nago1、nage2、nage3  
            代表烟草、纸、火柴
```

```
            if flag2 & flag3 then V(S1); /
```

```
            * 供纸和火柴
```



```
        else if flag1 & fiag3 then V(S2 ) ;    /
        else V(S3) ;                          /
until false ;
end
process 吸烟者 1
begin
repeat
P(S1) ;
取原料 ;
做香烟 ;
V(S) ;
吸香烟 ;

until false ;

process 吸烟者 2

begin
repeat

P (S2 ) ;

取原料 ;

做香烟 ;

V(S) ;

吸香烟 ;

until false ;

process 吸烟者 3

begin
repeat

P (S3 ) ;

取原料 ;

做香烟 ;

V ( S ) ;

吸香烟 ;
```

\* 供烟草和火柴  
\* 供烟草和纸

until false ;

coend .

( 3 ) 用管程。

TYPE mskesmoke=moonitor

VAR S, S1 ,S2 ,S3 : condition ;

flag1 , flag2, flag3 : boolean

DEFINE give , take1 , take2 , take3 ;

USE check , wait , signal , release ;

procedure give

begin

check ( IM ) ;

准备香烟原料 ;

if 桌面上有香烟原料 then wait( S , IM ) ;        把准备的香烟原料放桌上 ;

if fiag2 & flag3 then signal ( S1 ,IM        ) ;

if flag1 & flag3 then signal ( S2 ,IM ) ; else signal (S3 , IM ) ;

release ( IM ) ;

end

procedure take1

begin

check(IM):

if 桌上没有香烟原料 then wait ( S1 ,IM        ) ;

else 取原料 ;

signal ( S , IM ) ;

release ( IM ) ;

end

procedure take2

begin

check ( IM ) :

if 桌上没有香烟原料 then wait(S2,IM);

else 取原料 ;

signal ( S , IM ) ;

release ( IM ) ;

end

procedure take3

begin

check ( IM ) :

if 桌上没有香烟原料 then wait(S3,IM);

else 取原料

signal ( S ,IM ) ;

release ( IM ) ;

end

begin

flag1:=flag2:=flag3:=true;

end.

cobegin

{

process 供应者

begin

repeat

Call makesmoke.give();

”

until false ;

end

process 吸烟者 1

begin

repeat

Call makesmoke.take1() ;

做香烟 , 吸香烟 ;

until false ;

end

process 吸烟者 2

begin

repeat

Call makesmoke.take2() ;

做香烟 , 吸香烟 ;

until false ;

end

process 吸烟者 3

begin

repeat

Call makesmke.take3();

做香烟，吸香烟；

```
until false ;  
end  
}  
coend .
```

18、 如图所示，四个进程  $P_i$  ( $i=0, 3$ ) 和四个信箱  $M_j$  ( $j=0, 3$ )，进程间借助相邻信箱传递消息，即  $P_i$  每次从  $M_i$  中取一条消息，经加工后送入  $M_{(i+1) \bmod 4}$ ，其中  $M_0$ 、 $M_1$ 、 $M_2$ 、 $M_3$  可存放 3、3、2、2 个消息。初始状态下， $M_0$  装了三条消息，其余为空。试以  $P$ 、 $V$  为操作工具，写出  $P_i$  ( $i=0, 3$ ) 的同步工作算法

答：

```
var mutex1, mutex2, mutex3, mutex0 : semaphore;
```

```
mutex1 = mutex2 = mutex3 = mutex0 = 1;
```

```
empty0, empty1, empty2, empty3; semaphore;
```

```
empty := 0; empty1 := 3; empty2 := empty3 := 2;
```

```
full0, full1, full2, full3; semaphore;
```

```
full0 := 3; full1 := full2 := full3 := 0;
```

```
in0, in1, in2, in3, out0, out1, out2, out3; integer;
```

```
in0 := in1 := in2 := in3 := out0 := out1 := out2 := out3 := 0;
```

```
cobegin
```

```
{
```

```
process P0
```

```
begin
```

```
repeat
```

```
  P(full0);
```

```
  P(mutex0);
```

```
  从  $M_0[out_0]$  取一条消息；
```

```
   $out_0 := (out_0 + 1) \bmod 3$ ；
```

```
  V(mutex0);
```

```
  V(empty0)；
```

```
  加工消息；
```

```
  P(empty1)；
```

```
  P(mutex1)；
```

```
  消息已  $M_1[in_1]$ ;
```

In1:=(in1+1) mod 3;

V(mutex1) ;

V(full1 ) ;

until false ;

end

process P1

begin

repeat

P ( full1 ) ;

P ( mutex1 ) ;

从 M1[out1] 取一条消息 ;

Out1:=(out1+1) mod 3 ;

V(mutex1);

V(empty1);

加工消息 ;

P(empty2);

P(mutex2 ) ;

消息已 M2[in2];

In2:=(in2+1) mod 2;

V(mutex2 ) ;

v ( full2 ) ;

until false ;

end

process P2

begin

repeat

P(full2) ;

P(mutex2 ) ;

从 M2[out2] 取一条消息 ;

out2:=(out2 + 1 ) mod 2;

V(mutex2) ;

V(empty2) ;

加工消息 ;

P(empty3) ;

P(mutex3) ;

消息已 M3[in3];

```
in3:=(in3+1) mod 2 ;
```

```
V(mutex3) ;  
V(full3) ;  
until false ;  
end
```

```
process P3  
begin
```

```
repeat  
P(full3) ;  
P(mutex3) ;  
从 M3[out3] 取一条消息 ;
```

```
out3:=(out3+1)mod 2;
```

```
V (mutex3) ;  
V (empty3) ;  
加工消息 ;  
P ( empty0 ) ;  
P ( mutex0 ) ;  
消息已 MO[in0];
```

```
In0:=(in0+1) mod 3 ;
```

```
V(mutex0) ;  
V(full0) ;  
until false ;  
end
```

```
{
```

```
coend
```

19、有三组进程  $P_i$ 、 $Q_j$ 、 $R_k$ ，其中  $P_i$ 、 $Q_j$  构成一对生产者和消费者，共享一个由  $M_1$  个缓冲区构成的循环缓冲池  $buf_1$ 。 $Q_j$ 、 $R_k$  凡构成另一对生产者和消费者，共享一个由  $M_2$  个缓冲区构成的循环缓冲池  $buf_2$ 。如果  $P_i$  每次生产一个产品投入  $buf_1$ ， $Q_j$  每次从中取两个产品组装成一个后并投入  $buf_2$ ， $R_k$  每次从中取三个产品包装出厂。试用信号量和  $P$ 、 $V$  操作写出它们同步工作的程序。

答：

```
var mutex1 , mutex2 , mutex3 : semaphore;
```

```
empty1 , empty2 , full1 , full2 ; semaphore ;  
in1 , in2 , out1 , out2 : integer ; counter1 , counter2:integer ;
```

```
buffer1:array[0 , M1-1] of item ; buffer2:array[0 , M2-1]of item ;
```

```
empty1:=M1 ; empty:=M2;  
in1 : = in2 :=out1:=out2:=0 ; counter1:=counter2:=0 ;
```

```
fun1:=full2: = mutex1:=mutex2:=mutex3:=1;
```

```
cobegin  
{  
process Pi  
begin  
L1:  
P(empty1) ;  
P(mutex1 ) ;  
put an item into buffer [in1] ;  
in1:=(in1+1) mod M1 ;  
counter++;  
if counter1 = 2 then { counter1:=0;V(full1);}
```

```
V(mutex) ;  
goto L1;  
end
```

```
process Qj
```

```
begin  
L2:
```

```
P ( full2) ;  
P ( mutex1 ) ;
```

```
take an item from buffer1[out1];
```

```
out1:=(out1+1) mod M1;  
take an item from buffer1[out1] ;
```

```
out1:=(out1 + 1) mod M1 ;  
V ( mutex1 ) ;  
V ( empty1 ) ;  
V ( empty1 ) ;  
Process the products ;  
P ( emPty2) ;
```

```

P ( mutex2 ) ;
put an item into buffer2 [ in2 ] ;
in2:=( in2 + 1 ) mod M2 ;
counter2 + + ;
if counter2 = 3 then { counter2:=0 ;V( full2 ) ; }

```

```

V ( mutex2 ) ;
goto L2 ;

```

```

process Rk

```

```

begin L3 :

```

```

P ( full2 ) ;
P ( mutex2 ) ;
take an item from buffer2 [out2];

```

```

out2: = ( out2 + 1 ) mod M2 ;

```

```

take an item from buffer2 [out2] ;

```

```

out2:=( out2 + 1) mod M2 ;

```

```

take an item from buffer2 [out2];

```

```

out2:=(out2 + 1 ) mod M2 ;

```

```

v ( mutex2 ) ;
V ( empty2 ) ;
V ( empty2 ) ;
V ( empty2 ) ;
packet the products ;
goto L3 ;

```

```

end

```

```

}
coend

```

20 在一个实时系统中，有两个进程 P 和 Q，它们循环工作。P 每隔 1 秒由脉冲寄存器获得输入，并把它累计到整型变量 W 上，同时清除脉冲寄存器。Q 每隔 1 小时输出这个整型变量的内容并将它复位。系统提供了标准例程创建 PUT 和 OUT 卫 UT 供拍，提供了延时系统调用 Delay ( seconds )。试写出两个并发进程循环工作的算法。

答：



```

Var W ,V:integer;

Mutex:semaphore;

W:=0 ; V:=0 ;mutex:1;

cobegin {

process P

begin

repeat

P(mutex) ;

delay (1) ;

V= INPUT ;

W:=W + V ;

清除脉冲寄存器 ;

V (mutex) ; untile false ;

end

process Q

begin

repeat

P ( mutex ) ;

delay ( 60 ) ;

OUTPUT ( W ) ;

W : = 0 ;

V ( mutex ) ;

untile false ;

}

coend .

```

21 系统有同类资源  $m$  个，被  $n$  个进程共享，问：当  $m > n$  和  $m \leq n$  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  $m \leq n$  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？

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

$$\max(1) + \dots + \max(n) = (\text{need}(1) + \dots + \text{need}(n)) + (\text{alloc}(1) + \dots + \text{alloc}(n)) < m + n$$

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， $\text{alloc}(1) + \dots + \text{alloc}(n) = m$

另一方面所有进程将陷入无限等待状态。可以推出

$$\text{need}(1) + \dots + \text{need}(n) < n$$

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  $i$ ， $\text{need}(i) = 0$ ，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， $n \times m < m + n$  是成立的，

$$\text{等式变换 } n \times (m - 1) + n < n + m$$

$$\text{即 } n \times (m - 1) < m$$

$$\text{于是有 } n \times (m - 1) + 1 < m + 1$$

$$\text{或 } n \times (m - 1) + 1 \leq m$$

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。

23 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，直到驳船尾部 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。

```

var Sa , Sb : semaphore ;
Sa:=Sb:=1 ;
cobegin
{
    process    驳船
    begin
        P(Sa ) ;
        P(Sb ) ;
        船过桥 A 、 B ;
        V(Sa ) ;
        V(Sb ) ;
    end
end

```

```

    process  汽车
    begin
        P ( Sa ) ;
        P ( Sb ) ;
        车过桥 A 、 B ;
        V ( Sa ) ;
        V ( Sb ) ;
    end
}
coend

```

24 Jurassic 公园有一个恐龙博物馆和一个花园，有  $m$  个旅客租卫辆车，每辆车仅能乘一个一旅客。旅客在博物馆逛了一会，然后，排队乘坐旅行车，挡一辆车可用喊飞它载入一个旅客，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

```

var sc1 , sck , sc      , Kx,xc  , mutex : semaphore ;
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      , , )
    begin
        P ( sc1 ) ;          /          * 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
    end
end

```

```

V ( sck ) ; /*      释放一辆车 ， 即顾客找到一辆空车

P ( Kx ) ; /*      待游玩结束之后 ， 顾客等待下车

V ( sc1 ) ; /*      空车辆数加 1

End

Process 车辆 j(j=1,2,3 , )

Begin

L: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

车辆载着顾客开行到目的地 ；

V(xc); /*      允许顾客下车

Goto L;

End

coend

```

25 今有  $k$  个进程，它们的标号依次为  $1、2、\dots、k$ ，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  $K$ ，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

```

var waits , mutex :semaphore ;
numbersum:integer:=0 ;
wait:=0 ; mutex:=1 ;
cobegin
{
  process readeri ( var number:integer ; )
  begin
    P(mutex) ;
    L:if    numbersum+number < K then { V ( mutex ) ; P ( waits ) ; goto L ; }

    Then numbersum:=numbersum+number;

```

```
V (mutex ) ;  
Read file ;  
P(mutex ) ;  
numbersum:= numbersum-number ;  
V(waits ) ;  
V(mutex ) ;
```

2 ) 使用管程 :

```
TYPE sharefile = MONITOR  
VAR numbersum ,n : integer ;  
SF : condition ;  
DEFINE startread , endread ;  
USE wait , signal , check , release ;  
procedure startread ( var number : integer : ) ;  
begin  
check (IM ) ;  
L :if ( number + numbersum ) > K then {wait(SF,IM) ; goto L ; }  
  
Numbersum:=numbersum+number;  
release (IM ) ;  
end
```

```
procedure endread (var number:integer ; ) ;  
begin  
check(IM ) ;  
numbersum := numbersum - number ;  
signal ( SF , IM ) ;  
release ( IM ) ;  
end  
begin  
numbersum:=0  
end .  
main()  
{ cobegin  
process-i() ;  
coend  
}  
process-i()  
var number : integer ;  
begin  
number : = 进程读文件编号 ;  
startread(number);;  
read F ;
```

```
endread(number);  
end
```

26、设当前的系统状态如下：系统此时 Available=(1,1,2):

- 1 ) 计算各个进程还需要的资源数  $C_{ki} - A_{ki}$  ?
- ( 2 ) 系统是否处于安全状态，为什么？
- ( 3 ) P2 发出请求向量  $request_2(1, 0, 1)$  , 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  $req\_stl(1, 0, 1)$  , 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  $request_3(0, 0, 1)$  , 系统能把资源分给它吗？

答：( 1 ) P1 , P2 , P3 , P4 的  $C_{ki} . A_{ki}$  分别为：( 2 , 2 , 2 )、( 1 , 0 , 2 )、( 1 , 0 , 3 )、( 4 , 2 , 0 ) ( 4 ) 系统处于安全状态，存在安全序：P2 , P1 , P3 , P4  
( 5 ) 可以分配，存在安全序列： P2 , P1 , P3 , P4 .  
( 6 ) 不可以分配，资源不足。  
( 7 ) 不可以分配，不安全状态。

27 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：

系统此时处于安全状态吗？

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。

28 把死锁检测算法用于下面的数据，并请问：

Available=(1,0,2,0)

(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。

29 ) 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：

(1) 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。

30 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  $-N < X \text{ 产品数量} - Y \text{ 产品数量} < M$ 。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约束条件：

$-N < X \text{ 产品数量} - Y \text{ 产品数量}$

$X \text{ 产品数量} - Y \text{ 产品数量} < M$

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。

```
var mutex : semaphore = 1 /* 互斥信号量 */
```

```
sx, sy : semaphore;
```

```
sx = M-1 ; sy = N-1 ;
```

```
cobegin
```

```
{
```

```
process X
```

```
{ repeat
```

```
P(sx) ;
```

```
P ( mutex ) ;
```

```
将 X 产品入库 ;
```

```
V(mutex) ;
```

```
V ( sy ) ;
```



```
until false
}
```

```
process Y
{ repeat
P ( sy );
P ( mutex );
将 Y 产品入库 ;
V ( mutex );
V ( px );
until false
}
}
coend .
```

31 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： $A \text{ 零件数量} - B \text{ 零件数量} \leq n$ ； $B \text{ 零件数量} - A \text{ 零件数量} \leq n$ ； $A \text{ 零件数量} \leq m$ ； $B \text{ 零件数量} \leq m$ 。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环队列，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：

```
Var empty1,empty2,full1,full2:semaphore;
```

```
Mutex ,sa,sb:semaphore;
```

```
In1,in2,out1,out2:integer;
```

```
Buffer1,buffer2:array[0 , m-1]of item;
```

```
Empty1:=empty2:=m;
```

```
Sa:=sb:=n;
```

```
In1:=in2=out1:=out2:=0;
```

```
Cobegin
```

```
{
```

Process producerA

{repeat

P(empty1);

P(sa);

P(mutex);

Buffer1[in1]:=A    零件 ;

In1:=(in1+1)mod m;

V(mutex);

V(sb);

V(full1);

Untile false;

}

Process producer B

{repeat

P(empty2);

P(sb);

P(mutex);

Buffer2[in2]:=B    零件 ;

In2:=(in2+1)mod m;

V(mutex);

V(sa);

V(full2);

Untile false;

}

Process take

{repeat

P(full1);

P(full2);

P(mutex);

Take from buffer1[out1] and buffer2[out2]                      中的 A , B 零件 ;

Out1:=(out1+1)mod m;

Out2:=(out2+1)mod m;

V(mutex);

V(empty1);

V(empty2);

把 A 和 B 装配成产品 ;

Until false

}

}

Coend.

32 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  $B_1, B_2, \dots, B_{n_2}$  共用  $m$  个缓冲区，每个缓冲区只要写一次，但需要读  $n_2$  次。因此，可以把这一组缓冲区看成  $n_2$  组缓冲区，每个发送者需要同时写  $n_2$  组缓冲区中相应的  $n_2$  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  $\text{mutex}$  实现诸进程对缓冲区的互斥访问；两个信号量数组  $\text{empty}[n_2]$  和  $\text{full}[n_2]$  描述  $n_2$  组缓冲区的使用情况。其同步关系描述如下：

```
var mutex, empty[n2], full[n2]:semaphore;
```

```
i:integer; mutex=1;
```

```
for(i=0;i<=n2-1;i++)
```

```
{
```

```
empty[i]=m;
```

```
Full[i]=0;
```

```
}
```

```
main ( )
```

```
{
```

```
cobegin
```

```
A1 ( ) ;
```

```
A2 ( ) ;
```

```
,
```

```
An1 ( ) ;
```

```
B1 ( ) ;
```

```
B2 ( ) ;
```

```
,
```

```
Bn2 ( ) ;
```

```
coend
```

```
send ( ) / * 进程 Ai 发送消息 * /
```

```
{ int i ;
```

```
for ( i=0;i<=n2-1;i++ ) ;
```

```
P(empty[i]);
```

```
P (mutex ) ;
```

```
将消息放入缓冲区 ;
```

```
V ( mutex ) ;
```

```
for(i=0 ; i<=n2-1;i++)
```

```
V(full[i]);
```

```
}
```

```
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 ( empty[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(1)
{
,
send ( ) ;
,
}
}
Bi ( ) / * 接收进程 B1 , B2 , , BnZ 的程序类似 , 这里给出进程 Bi 描述 * /

{
while(i)
(
,
receive ( i ) ;
,

}

}

```

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，且已知这 4 个进程均按以下顺序使用设备：

一申请 R1 一申请 R2 一申请 R1 ~ 释放 R1 一释放 R2 一释放 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。

34 如图所示，左右两队杂技演员过独木桥，为了保证安全，请用 PV 操作和信号量来解决过独木桥问题。只要桥上无人，则允许一方的人过桥，待一方的人全部过完后，另一方的人才允许过桥。

答：

```
var wait, mutex1, mutex2, bridge1, bridge2 : semaphore ;
mutex1:=mutex2:=bridge1:=bridge2:=1;wait:=0;
counter1, counter2 : integer ;
cobegin
{
process P 左                                process P 右
begin
P ( mutex1 ) ;                                P ( mutex2 ) ;

Count1 ++;                                count2 ++;

if count1 = 1 then P( wait ) ;                if count2 = 1 then P( wait ) ;

V ( mutex1 ) ;                                V( mutex2) ;

P(bridge1) ;                                P ( bridge2 ) ;

过独木桥 ;                                过独木桥 ;
```

```

V ( bridge1) ;                V( bridge2 ) ;

P ( mutex1) ;                P ( mutex2 ) ;

Count1-- ;                    count2-- ;

if count1 = 0 then V(wait) ;    if count2 = 0 then P
(wait) ;

V ( mutex1) ;                V (mutex2) ;

end ;                        end ;
} coend

```

35 修改读者—写者的同步算法，使它对写者优先，即一旦有写者到达，后续的读者必须等待，而无论是否有读者在读文件。（ 1 ）用信号量和 P 、 v 操作实现；（ 2 ）用管程实现。

答：（ 1 ）用信号量和 P、V 操作实现

为了提高写者的优先级，增加了一个信号量 S，用于在写进程到达后封锁后续的读者。其控制流程如下：

```

Var rmutex,wmutex,s:semaphore;

```

```

Rmutex=1;wmutex=1;s=1;

```

```

Count:integer:=0;

```

```

Main()

```

```

{cobegin

```

```

    Reader();

```

```

    Writer();

```

```

Coend

```

```

}

```

```

Reader()

```

```

    Begin

```

```

        While(1)

```

```
{ P(s);

P(rmutex);

If(count==0) P(wmutex);

Count++;

V(rmutex);

V(s);

读文件 ;

P(rmutex);

Count--;

If (count==0) v(wmutex);

V(rmutex);

}

Writer()

Begin

While(1)

{

P(s);

P(wmutex);

写文件 ;

V(wmutex);

V(s);

}

End.
```

(2) 用管程实现



TYPE read-write=monitor

Var rc,wc:integer;

R,W:condition;

DEPINE start-read , end-read , start-riter , end-writer;

USE wait , signal , check , release ;

procedure start-read;

begin

check ( IM ) ;

if wc > 0 then wait ( R ,IM ) ;

rc:=rc + 1;

signal ( R , IM ) ;

release ( IM ) ;

end ;

procedure end-read ;

begin

check ( IM ) ;

rc:=rc-1 ;

If rc=0 then signal ( W , IM ) ;

release ( IM ) ;

end ;

procedure start-write ;

begin

check ( IM ) ;

wc:=wc + 1 ;

if rc > 0 or wc > 1 then wait ( W , IM ) :

release ( IM ) ;

end ;

procedure end-write ;

begin

check ( IM ) ;

wc:=wc-1 :

if wc > 0 then signal ( W , IM ) ;

else signal ( R , IM ) ;

release ( IM ) ;

end ;

begin

```
rc:=0; wc:=0 ; R:=0 ; W:=0 ;  
end .
```

```
Cobegin {
```

```
process P1
```

```
begin
```

```
”
```

```
call read-writer.start-read;
```

```
”
```

```
Read;
```

```
call read-riter.end-read ;
```

```
end ;
```

```
process P2
```

```
begin
```

```
”
```

```
Call read-writer.start-writer;
```

```
”
```

```
Write;
```

```
”
```

```
Call read-writer.end-write;
```

```
”
```

```
End;
```

```
}
```

```
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（或称进程资源图）。

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。

```
var F1, F2 : ARRAY [ 0 , 9 ] of item;
```

```
SP1 , SP2 , SI1 , SI2:seMaphore ;
```

```
in1 , in2 , out1 , out2 : integer ;
```

```
in1:=0;in2:=0;out1:=0 ; out2:=0 ;
```

```
SP1:=10;SP2:=10;SI1:=0;SI2:=0;
```

```
Main()
```

```
{cobegin
```

```
Producer1();
```

```
Producer2();
```

```
Installer()
```

Coend

}

Process producer1()

Begin

While(true)

{

Produce A 零件 ;

P(SP1);

F1[in1]:A;

In1:=(in1+1) mod 10

V(SI1);

}

End

Process producer2()

Begin

While(true)

{

Produce B 零件 ;

P(SP2);

F2(in2):=B;

In2:=(in2+1) mod 10

V(SI2);

}

End

Process installer()

Var product:item;

Begin

While(true)

{ p(SI1);

Product1:=F1[out1];

Out1:=(out1+1) mod 10;

V(SP1);

P(SI2);

Product2:=F2[out2];

Out2:=(out2+1) mod 10;

V(SP2);

组装产品 ;

}

End

TYPE produceprodut=monitor

VAR F1 , F2 : ARRAY [ 0 , 9 ] of item;

SP1 , SP2 , SG1 , SG2:semaphore;

SP1\_count1,SP2 count2 , SG1\_count,SG2\_count:integer;

In1, in2 , out1 , out2:=integer ;

inc1 , inc2 : integer ;

DEFINE put1 , put2 , get :

USE wait,signal;

procedure put1( A );

begin

if inc1=10 then wait ( SP1 , SP1\_count , IM );

Inc1:=inc1 + 1 ;

F1[in1]:= A ;

in1:=(in1 + 1 ) MOD 10

```

signal ( SG1 , SG1_count , IM ) ;
end :
procedure put2 ( B ) :
begin
if inc2 =10 then wait ( SP2 , SP2_count , IM );

Inc2 :=inc2 + 1 ;
F2 [in2]:=B;
in2:=(in2 + 1 ) MOD 10
signal ( SG2 , SG2_count , IM ) ;
end ;
procedure get ( A , B ) ;
begin
if inc1=0 then wait ( SG1 , SG1_count , IM ) ;

if inc2=0 then wait ( SG2 , SG2_count , IM ) ;

inc1:=inc1-1 ;
inc2:=inc2-1;
A:F1[out1];
out1:=(out1 + 1 ) MOD 10
B:=F2[out2];
Out2 :=(out2 + 1 ) MOD 10
signal ( SP1 , SP1_count , IM ) ;
signal ( SP2 , SP2_count , IM ) ;
end ;

begin
in1:=0 ;in2:=0;out1:=0;out2:=0;inc1:=0;inc2:=0 ;

SP1:=0;SP2:=0;SG1:=0;SG2:=0;
end.

cobegin
{
process Produce1

begin

while(true)

{produce A 零件 ;

P(IM.mutex);

```

Call produceprodut.put1(A);

If IM.next>0 then V(IM.next);

Else V(IM,mutex);

}

End;

Process Produce2

Begin

While(true)

{produce B 零件;

P(IM.mutex);

Call produceprodut.put2(B);

If (IM.next>0 then V(IM.next);

Else V(IM,mutex);

}

Process consume

Begin

While(true)

{

P(IM.mutex);

Call produceprodut.get(A,B);

If IM.next>0 then V(IM.next);

Else V(IM,mutex);

组装产品 ;

}

End;

}

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。

Var

plate ARRAY [ 0 , 1] of ( apple , orange ) ;

flag0 , flag1:=boolean ;

mutex : semaphore ;

sp : semaphore; / \* 盘子里可以放几个水果  
\* /

sg1 , sg2 : semaphore ; / \* 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 son  
process father begin  
begin L3 : P (sg1 ) ;

L1 : 削一个苹果 ; P( mutex ) ;  
P ( sp ) ; if ( flag0&flag1[0]== 桔  
子 ) then

If(flag0==false) then  
else{x:=plate[1];flag1:=false;}  
{ plate[0]: = 苹果 ; flag1:=true;} v(mutex);

else {plate[1]:= 苹果 ; flag1:=true;} V(sp) ;  
v ( mutex ); 吃桔子 ;



```

v(sg2)                                goto L3;
goto L1 ;                             end;
end ;
process mother                        process daughter
begin                                begin
L2 : 剥一个桔子 ;                    L4 : P ( 592 ) :

P ( sp ) ;                            P ( mutex )

P ( mutex ) ;                        if ( flag0 & plate
[0]= = 苹果 ) then
if ( flag0==false ) then              {x:=plate [01];
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 ;
goto L2 ;                            goto L4;
end ;                                end ;
coend .

```

( 2 ) 用管程 .

TYPE FMSD = MONITOR

VAR plate ARRAY [ 0 , 1 ] of ( apple , orange ) ;

Count:integer ; flag0 , flag1:boolean ;

SP ,SS , SD : codition ;

DEFFINE put,get ;

USE wait,signal , check , release ;

procedure put(var fruit:( apple , orange ) ) ;

begin

check(IM ) ;

if ( count= = 2 ) then wait(SP , IM ) ;

else{if(flag0==false) then

{plate[0]:=fruit; flag0:=true;}

Else{plate[1]:=fruit;flag1:=true;}

```
Count:=count+1;

If(fruit==orange) then signal(ss,IM);

Else signal(SD,IM);

}

Release(IM);

End;

Procedure get(varfruit:(apple,orange),x:plate);

Begin

Check(IM);

If (count==0) or plate <>fruit

Then begin

If(fruit==orange) then wait(SS,IM);

Else wait(SD,IM);

End;

Count:=count-1;

If(flag0&plate[0]==fruit) then

{x:=plate[0];flag0:=false;}

Else{x:=plate[1];flag1:=false;}

Signal(SP,IM);

Release(IM);

End;

Begin

Count:=0;flag0:=false;flag1:=false; SP:=0;ss:=0;sd:=0;

Plate[0]:plate[1]:=null;
```

End;

Main()

{cobegin

Process father

Begin

While(1)

{ 准备好苹果 ;

Call FMSD.put(apple);

”

}

End;

Process mother

Begin

While(1)

{

准备好桔子 ;

Call FMSD.put(orange);

”

}

End;

Process son

Begin

While(1)

{call FMSD.get(orange,x);

吃取到的桔子；

”

}

End;

Process daughter

Begin

While(1)

{

Call FMSD.get(apple,x);

吃取到的苹果；

”

}

End;

}

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 buf : ARRAY [ 0 , 8 ] of integer ;

count,getptr , putptr : integer ;

count:=0 ; getptr:=0;putptr:=0 ;

S1 , S2 , SPUT , SGET ; semaphore ;

S1:=1 ; S2 :=1 ; SPUT : = 1 ; SGET :=0 ;

main ( )

{ cobegin

producer-i( ) ;

consumer-j( ) ;

coend

}

```

process producer-i
begin
L1 : 生产 3 个整数 ;

P(SPUT);

P(S1);

Buf[putptr]:= 整数 1;

Putptr:=(putptr+1)mod 9;

Buf[putptr]: = 整数 2 ;

putptr :=(puttr+1 ) MOD 9

buf[putptr]:= 整数 3 ;

putptr:=(putptr+1) MOD 9;

V ( SGET ) ;

v ( SGET ) ;
v ( SGET ) :
v ( S1 ) ;
goto L1
end

process consumer-j
var y:integer ;
begin
L2:P(SGET ) ;
P ( S2 ) ;
y=buf[getptr] ;
getptr:=(getptr + 1) MOD 9 ;

count:=count + 1;
if count= 3 then
begin
count:=0 ;
V ( SPUT ) ;
end
V ( S2 ) ;

consume the 整数 y;

```

goto L2 ;

end

( 2 ) 管程。

TYPE get-put = MONITOR

VAR buf ARRAY [ 0 , 8] of integer ; count , getptr , putptr:integer ;

SP , SG ; condition

DEFINE put,get ;

USE wait ,signal , check , release ;

Procedure put(var a1 , a2 , a3 :integer ; ) ;

begin

check(IM ) ;

if ( coun>6 ) then wait(SP , IM ) ;

count:count + 3 ;

buf[putptr]:=a1 ;

putptr(put1+1 ) MOD 9;

buf [putptr]:=a2 ;

putptr:=(putptr+1) MOD 9 ;

buf[putptr]:=a3;

putptr:=(putptr+1) MOD 9 ;

signal(SG,IM);

release(IM ) ;

end ;

procedure get (b);

begin

check(IM);

if ( count==0 ) then wait(SG,IM ) ;

b:buf[getptr] ;

getptr:=(getptr + 1 ) MOD 9 ;

count :=count + 1 ;

if count < 7 then signal ( SG,IM ) ;

else if count > 0 then signal ( SG,IM ) ;

```

release ( IM ) ;
end ;
begin
count:=0 ; getptr:=0;putptr:=0 ;

SP:=0;SG:=0;

End;

cobegin
{
process producer-i
begin
L1 : 生产 3 个整数 ;
Call get-put.put(a1, a2 , a3 ) ;

goto L1
end

process consumr-j
var y:integer ;
begin
L2 : call get-put.get(b)

consume the 整数 b ;

goto L2;
end
}
coend

```

40 设有三个进程 P 、 Q 、 R 共享一个缓冲区， P 进程负责循环地从磁带机读入一批数据并放入缓冲区， Q 进程负责循环地从缓冲区取出 P 进程放入的数据进行加工处理并把结果放入缓冲区， R 进程负责循环地从缓冲区读出 Q 进程放入的数据并在打印机上打出。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 Sp , Sq , Sr : semaphore;

Buf : integer;

SP:=1;SP:=Sr:=0;

Cobegin

{process P

Begin

Repeat

从磁带读入数据；

P(SP);

Buf:=data;

V(sq);

Until false;

End

Process Q

Begin

Repeat

P(sq);

Data:=buf;

加工处理 data;

Buf:=data;

V(Sr);

Until false;

End

Process R

Begin

Repeat

P(Sr);

Data:=buf;

V(sp);

打印数据



Until false;

End

}

(2)管程

TYPE PQR=MONITOR

VAR buf:integer;

SP,SQ,SR:condition;

Turn:{p,q,r};

DEFINE PPUT,QGET,QPUT,RGET;

USE wait,signal,check,release;

Procedure PPUT(var data:integer;);

Begin

Check(IM);

If turn!=p then wait (sp,IM);

Turn:=q;

Buf:=data;

Signal(SQ,IM);

Release(IM);

End

Process QGET(var data:integer;);

Begin

Check(IM);

If turn !=q then wait(SQ,IM)

Data:buf

Release(IM);

End

Procedure QPUT(var data:integer;);

Begin

Check(IM);

Turn:=r;

Buf:=data;

Signal(SR,IM);

Release(IM);

End

Procedure RGET(var data:integer;);

Begin

Check(IM);

If turn !=r then wait(SR,IM);

Turn:=p;

Data:buf

Signal(SP,IM);

Release(IM);

End

Begin

Sp:=0;SQ:=0;SR:=0;turn:=p;

End

Main()

{cobegin

Process P

X:=integer;

Begin

LP:从文件读入一个数据到 X;

PPUT(X);

Goto LP;

End

Process Q

X:integer;

Begin

LQ:QGET(x);

加工处理 X;

QPUT(x);

Goto LQ;

End

Process R

X:=integer;

Begin

LR:RGET(X);

打印 X;

Goto LR;

End

}

Coend

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 exclusinn)、 “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。

```
Program attemp;  
  
Var c1,c2:integer;  
  
Procedure p1;      (/*      对第一个进程 P1*/)   
  
Begin  
  
Repeat  
  
Remain section 1;  
  
Repet  
  
C1:=1-c2;  
  
Until c2<>0;  
  
Critical section;  (/*      临界区 */)   
  
C1:=1;  
  
Until false  
  
End;  
  
Procedure p2;      (/*      对 另一个进程 p2*/)   
  
Begin  
  
Repet  
  
Remain section 2;  
  
Repeat  
  
C2:=1-c1  
  
Until c1<>0;  
  
Critical section; (/*      临界区 */)   
  
C2:=1  
  
Until false  
  
End;
```

```

Begin      (/*      主程序 */)

C1:=1;

C2:=1;

Cobegin

P1;P2      (/*      两进程 P1,P2 开始执行 */)

Coend

End

```

答：( 1 ) 互斥

已知  $c_1$  和  $c_2$  的初值为 1，若进程 P1 执行到  $c_1 := 1 - c_2$  时，进程 P2 也同时执行  $c_2 := 1 - c_1$ 。这样一来， $c_1$  和  $c_2$  的值都变为 0，接着再各自执行，repeat---until 循环语句  $c_1 := 1 - c_2$  和  $c_2 := 1 - c_1$  时， $c_1$  和  $c_2$  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  $c_1 := 1 - c_2$ ，由于  $c_2$  的初值为 1，这使得  $c_1$  的值变为 0 但  $c_2$  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  $c_1 := 1$ ，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  $c_1 := 1$ ，这使得进程 P2 因  $c_1$  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 分析下列算法是否正确，为什么？

```

repeat

key:=true;

repeat

swap ( lock , key ) :

until key=false;

```

Critical section        ( / \* 临界区 \*/ )

Lock:=false;

Other code ;

Until false;

答：由于 lock 的初值未定，如果它的值 false，则可通过 swap 实现上锁操作。但如果 lock 的初值为 true，那么，进程会永远等待而进不了临界区。

43 以下并发执行的程序，仅当数据装入寄存器后才能加        1

Const n =50 ;

var tally :integer ;

procedure total ( )

var count    : integer ;

Begin

For count:=1 to n do tally:=tally+1

End;

Begin (/\*main program\*/)

Tally:=0;

Cobegin

Total();total()

Coend;

Writeln(tally);

End.

给出该并发程序输出的 tally 值的上限和下限。

答：tally 值的上限和下限为 100 和 50。

44 举例说明下列算法不能解决互斥问题。

var balocked    : array[ 0 , 1] of boolean ;

```

turn:0 , 1;
procedure P[id:integer];
begin

repeat
blocked[id]:=true;
while turn = id do
begin
while blocked [1-id] do Skip;

turn: = id ;
end;
{critical section }
blocked[id]: = false :
{remainder }
until false

end;
begin
blocked [ 0 ]: blocked[1]:=false ;

turn:=0;
cobegin
P[0] ;P[1]
coend ;
end.

```

答：为方便描述，把程序语句进行编号：

```

Blocked[id]:=true;
while turn = id do
begin
while blocked[1-id] do skip;

Turn:=id;

End;

```

假设 id=0，则 1-id=1，并且 turn = 1。当进程 P[id] 先执行置 blocked[id]=true：接着执行时，因为 turn = id 而进入到执行。此时，因 blocked[1-id] 为 false（初值），故在上不做空操作而打算去做。麻烦的事情发生了，如果在 P[id] 执行之前，系统又调度执行 P[1-id]，而 P[1-id] 在执行了置 blocked[1-id]=true 之后，在执行时，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，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。

45 现有三个生产者 P1、P2、P3，他们都要生产水，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

VAR S, S1, S2, S3 : condition ;

container:item ;

DEFINE give, produce1, produce2, produce3 ;

USE check, wait, signal, release ;

procedure give

begin

Check ( IM ) ;

take raw material ;

if container = null then wait ( S, IM ) ;

else container := raw material ;

if ( container = 桔子精 then signal ( s1, IM ) ;

else if ( container = 糖 then signal ( S2, IM ) ;

else signal ( S3, IM ) ;

release ( IM ) ;

end

procedure produce1

begin

check ( IM ) ;

if ( container = 桔子精 then wait ( s1, IM ) ;

else { take the 桔子精 from container ; 做桔子水 ; }

signal ( S, IM ) ;

release ( IM ) ;

end

procedure produce2

begin

check(IM);

IF(CONTAINER) 糖 then wait(S2,IM);



Else{take the 糖 from container; 做橘子水 ; }

Signal(S,IM);

Release(IM);

End

Procrdure produce3

Begin

Check(IM);

If(container 水 then wait(S3,IM);

Else{take the 水 from container; 做橘子水 ;}

Signal(S,IM);

Release(IM);

End

Begin

Container{ 糖 , 水 , 橘子精 };

End

Cobegin

{

Process 供应商

Begin

Repeat

,

Call makedrink.give();

,

Until false;

End

Process P1

Begin repeat

,

Call makedrink.produce1();

,

Until false;

End

Process P2

Begin

Repeat

,

Call makedrink.produce2();

,

Until false;

End

Process P3

Begin

Repeat

,

Call makedrink.produce3();

,

Until false;

End

}

Coend.

## 2) 信号量与 P、V 操作

Var S,S1,S2,S3:=semaphore;

S:=1,S1:=S2:=S3:=0;

Container{ 糖 , 水 , 橘子精 };

Cobegin

{ process 供应商

Begin

Repeat

P(s);

Take raw material into container;

If (container)= 橘子精 then V(S1);

Else if (container)= 糖 then V(s2);

Else V(s3);

Until false;

End

Process P1

Begin

Repeat

P(S1);

Take the 橘子精 from container;

V(s);

做橘子水 ;

Until false;

End

Process P2

Begin

Repeat

P(s2);

Take the 糖 from container;

V(s);

做橘子水；

Until false;

End

process P3

begin

repeat

P ( S3 ) ;

take the 水 from container;

V ( S ) ;

做桔子水；

until false；

end

}

coend .

46 有一材料保管员，他保管纸和笔若干。有 A、B 两组学生，A 组学生每人都备有纸，B 组学生每人都备有笔。任一学生只要能得到其他一种材料就可以写信。有一个可以放一张纸或一支笔的小盒，当小盒中无物品时，保管员就可任意放一张纸或一支笔供学生取用，每次允许一个学生从中取出自己所需的材料，当学生从盒中取走材料后允许保管员再存放一件材料，请用：1) 信号量与 P、v 操作，2) 管程，写出他们并发执行时能正确工作的程序。

答：1) 信号量与 P、v 操作。

var s, Sa, Sb, mutexA, mutexB : semaphore;

s := mutexA := mutexB := 1; sa := sb := 0;

box : ( Paper, pen );

cobegin

```

{
process  保管员
begin
repeat
P ( S ) ;
take a material intobox ;
if ( box ) = Paper then V ( Sa ) ;
else V( Sb ) ;
untile false ;
end

```

```

Process A  组学生
begin
repeat
P ( Sa ) ;
P ( mutexa ) ;
take the pen from box ;
V ( mutexa ) ;
V ( S ) ;
write a letter;
untile false ;
end

```

```

Process B  组学生
begin
repeat
P ( Sb ) ;
P ( mutexb ) ;
take the paper from box ;

V ( mutexb ) ;
V ( S ) ;
wnte a letter ;
untile false ;
end
}
Coend .

```

2 ) 管程。

```

TYPE paper&pen = monitor
VARS , S1 , S2 : condition ;
box : { paper.pen , null }
DEFINE put , get1 , get2 ;
USE check , wait , signal , release ;

```

```

procedure put
begin
Check ( IM ) ;
take a material ;
if box      null then wait ( S ,IM ) ;
else box := material ;
if ( box) = Pen then signal ( S1 , IM ) ;
else signal (S2 , IM ) ;
release ( IM ) ;
end
procrdure get1
begin
check ( IM ) ;
if ( box ) = null or ( box      )   pen then wait ( S1 , IM ) ;

else {take the Pen from box ; }
signal ( S , IM ) ;
release ( IM ) ;
end
procrdure get2
begin
check ( IM ) ;
if ( box ) = null or ( box      )   paper then wait ( S2 , IM ) ; else { take
the paper from box ; }
Signal ( S ,IM );

release ( IM ) ;

end

begin
box := null ;
end
cobegin
Process  保管员
begin
L1 : Callp paper&Pen.put      ) ; goto L1
end
Process A  组学生

begin {

L2 : call paper&pen.get ( )

```

```

写信 ;
goto L2 ;

end

process B  组学生
begin
L3 : call paper&pen.get ( )      写信 ;
goto L3 ;
end

coend

```

47 进程 A 向缓冲区 buffer 发消息，每当发出一条消息后，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。试写出：(1) 用信号量和 P、v 操作，(2) monitor，写出它们同步工作的程序。

答：(1) 用信号量和 P、v 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。

```
var Sa, Sb, Sc, Sd : semaphore ;
```

```
Sa := 3 ; Sb := Sc := Sd := 0 ;
```

```
cobegin
```

```
{ process A
```

```
begin
```

```
repeat ;
```

```
P ( Sa ) ;
```

```
P ( Sa ) ;
```

```
P ( Sa ) ;
```

```
Send message to buffer ;
```

```
V ( Sb ) ;
```

```
V ( Sc ) ;
```

```
V ( Sd ) ;
```

```
until false ;
```

```
end
```

```
process B
```

```
begin
```

```
repeat
```

```
P ( sb ) ;
```

```
receive the message from buffer ;
```

```
V ( Sa ) ;
```

```
until false ;
```

```
end
```

```

Process C
begin
repeat
P ( Sc ) ;
receive the message from buffer ;

V ( Sa ) ;
until false ;
end

process D
begin
repeat
P ( Sd ) ;
receive the message from buffer ;

V ( Sa ) ;
until false ;
end
}
coend
( 2 ) monitor .
TYPE send&receive=monitor
VAR SSb , SSc , SSd , Sb , Sc , Sd : selpaphore ;

SSb_count , SSc_pount , SSd_count : integer;

Sb_count , Sc_count , Sd_count      : integer;

fiagb , fiagc , fiagd : Boolean ;
buffer : message ;
DEFINE sendmes receiveb receivec received ;

USE wait , signal ;
procedure sendmes
begin
if flagb then wait ( sb , Sb_count      , IM ) ;

if flagc then wait ( Sc , Sc_count , IM ) ;

if flagd then 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 ) ;  
end
```

```
procedure receiveb  
begin  
if flagb = false then wait ( SSb , SSb_count , IM ) ;  
  
else flagb : = false ;  
signal ( Sb , Sb_count , IM ) ;  
end
```

```
procedure receivec  
begin  
if flagc = false then wait ( SSc , SSc_count , IM ) ;  
  
else flagb : = false ;  
signal ( Sc , Sc_count , IM ) ;  
release ( IM ) ;  
end
```

```
procedure received  
begin  
check ( IM ) ;  
if flag=false then wait ( SSd , IM ) ;
```

```
else flagb : = false ;  
signal ( Sd , Sd_count , IM ) ;  
release ( IM ) ;  
end  
begin  
flagb : = flagc : = flagd : = false ;  
end
```

```
cobegin  
{ process A  
begin  
repeat  
produce a message ;  
P ( IM.mutex ) ;  
Call send&receive.sendmes() ;
```

```
If IM.next > O then V ( IM.next ) ;
```

```
Else V ( IM.mutex ) ;
```

```
,
```

```

until false ;

end
process B

begin
repeat
P ( IM . mutex ) ;
Call send&receive . receiveb();

If IM . next > 0 then V ( IM . next ) ;

Else V ( IM . mutex ) ;

,

until false ;
end

process C
begin
repeat
P ( IM . mutex ) ;
Call send&receive . receiveco ; If IM . next > 0 thenV ( IM . next ) ;

elseV ( IM . mutex ) ;

,

until false ;
end

processD
begin
repeat
P ( IM . next ) ;
Call send&receive . receivedo ; If IM . next > 0 thenV      ( 加 . next ) ;

elseV ( IM . mutex ) ;
until false ;
end
}
Coend

```

48 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

```

```

var headpos : integer ;
direction ( up , down ) ;
busy : boolean ;
S : array [0      , 99]of condition ;
DEFINE request , return ;
USE wait , signal , check , release ;
procedure request ( var dest : integer ) ;
begin
check ( IM ) ;

if busy then wait ( S[dest] , IM ) ;
busy :   = true ;
if ( headpos < dest ) or(headpos = dest&direction = up )
then direction : = up ;
else direction : = down ;
headpos : = = dest ;
release ( IM ) ;
end

procedure return
vari : integer ;
begin
check ( IM ) ;
busy : = false ;
if direction   = = up      /      * uP 为向里方向 , 即柱面号大的方向小   en begin*/

i : = headpos ;
while ( i < 200 & S [ i ] = 0 ) do i : = i + 1 ;
if i < 200 then Signal ( S [i] , IM ) ;
else begin           / * down           为向外方向 , 即柱面号小的方向   i : 角
eadPos ;*/
while ( i      0 & S [i]   = 0 ) do i : = i-1;
if i      0 then signal ( S [i] , IM ) ;
end
end
else begin           / * down           为向外方向 , 即柱面号小的方向   i : = h
eadPos ;
while ( i > 0 & S [ 1]= 0 ) do i : = i -1 ;
ifi      0 then signal ( S [ i]           , IM ) ;
else begin           /           * 即为向里方向 , 即柱面号大的方向   i : = =
headPos ;
while ( i < 200 & S [ i ] = 0 ) do i : = i + 1 ;
if i < 200 then signal ( S [ 1 ] , IM ) ;
end
end

```

```

release ( IM ) ;
begin
headpos := 0 ;
direction :      = up ;
busy := false ;
S := 0 ;
end .
main()
{ cobegin
process visit

var k : integer ;

begin

,

call diskschedul.Request(k) ;

,

访问第 k 个柱面 ;

,

call diskschedul . Return ;

,

end

coend .

```

49 有 P1 、 P2s 、 P3 三个进程共享一个表格 F，P1对 F 只读不写，P2 对 F 只写不读，P3 对 F 先读后写。进程可同时读 F，但有进程写时，其他进程不能读和写。用（ 1 ）信号量和 P、v 操作，（ 2 ）管程编写三进程能正确工作的程序。

答：（ 1 ）信号量和 P、v 操作。

这是读—写者问题的变种。其中，P3 既是读者又是写者。读者与写者之间需要互斥，写者与写者之间需要互斥，为提高进程运行的并发性，可让读者尽量优先。

```

var rmutex , wmutex : semaphore ;
rmutex := wmutex := 1 ;
count : integer ; count := 0 ;
cobegin
{

```

```

process P1
begin
repeat
P (rmutex ) ;
count := count + 1 ;
if count= 1 then P( wmutex ) ;
V ( rmutex ) ;
Read F ;
P ( rmutex ) ;
count := count - 1 ;
if count=0 then V ( wmutex ) ;
V ( rmutex ) ;
until false ;
end

```

```

process P2
begin
repeat
P ( wmutex ) ;
Write F ;
V ( wmutex);

```

```

until false ;

```

```

process P3
begin
repeat
P ( rmutex ) ;
count := count + 1 ;
if count=1 then P ( wmutex ) ;

```

```

V ( rmutex ) ;
Read F ;
P ( rmutex ) ;
coUnt := count-1 ;
if count = 0 then V( wmutex );

```

```

V ( rmutex ) ;
P ( wmutex ) ;
Write F ;
V(wmutex ) ;
until false ;
end

```

```

}

```

coend

( 2 ) 管程。  
见课本读者写者问题的解。

50、 现有 100 名毕业生去甲、乙两公司求职，两公司合用一间接待室，其中甲公  
司招收 10 人，乙公司准备招收 10 人，招完为止。两公司各有一位人事主管在  
接待毕业生，每位人事主管每次只可接待一人，其他毕业生在接待室外排成一个  
队伍等待。试用信号量和 P、v 操作实现人员招聘过程。

答：由于毕业生仅排成一队，故用如图的一个队列数据结构表示。在队列中不含  
甲、乙公司

A	B	A	A	B	Sm	B	Sn	A	,						
					A		B								

都接待过的毕业生和已被录用的毕业生。只含标识为 A（被甲接待过）或只含  
标识为 B（被乙接待过）及无标识的毕业生队列。此外，sm 和 Sn 分别为队列  
中甲、乙正在面试的毕业生 i ( i = 1, 2 , , , 100 ) 标识、即此刻另一方  
不得面试该毕业生 i 。

K1 和 K2 为甲、乙所录取的毕业生数，C1、C2 为互斥信号量。注意，如果甲录  
取了一人，且该生没有被乙面试的话，则乙面试的毕业生将减 1。办法是：如  
果甲录取了一人，且该生没有被乙面试可把乙的面试计数器 C2 加 1（相当于乙  
已面试了他），从而，保证乙面试的人数值为 100。反之对甲亦然。

var Sa, Sb, mutex : semaphore ;  
Sa := Sb := mutex := 1 ;  
C1, C2, K1, K2 : integer ;  
C1 := C2 := K1 := K2 := 0 ;

cobegin  
{  
process 甲公司  
begin  
L1: P ( mutex ) ;  
P ( Sa ) ;  
C1 := C1 + 1 ;  
V ( Sa ) ;  
If C1 = 100 then  
{ 从标识为 B 且不为 Sn 或无标识的毕业生队列中选第 i 个学生，将学生 i 标  
识为 A 和 Sm}  
V ( mutex ) ;  
面试 ;  
P ( mutex ) ;  
if 合格 then

```

{ K1 := K1 + 1 ;
if 学生 i 的标识不含 B then
{ P ( Sb ) ;
C2 := C2 + 1 ;
V ( Sb ) ;
将学生 i 从队列摘除 ;
}
else 将学生 i 从队列摘除 ;

else
if 学生 i 的标识含 B then 将学生 i 从队列摘除 ;

else 取消学生 i 的 Sm标识 ;
V ( mutex ) ;
If ( K1 < 10 ) & ( C2 < 100 ) then goto L1 ;
}

process 乙公司

begin

L2 : P ( mutex ) ;

P ( Sb ) ;

C2 := C2 + 1 ;

V ( Sb ) ;
if C2 = 100 then
{ 从标识为 A 且不为 sm 或无标识的毕业生队列中选第 i 个学生 , 将学生 i 标识为 B 和 Sn}

V ( mutex ) ;
面试 ;
P ( mutex ) ;
if 合格 then
{ K2 := K2 + 1 ;
if 学生 i 的标识不含 A then

{

P(Sa)

C1 := C1 + 1 ;
V ( Sa ) ;
将学生 i 从队列摘除 ;
}
}
}

```

```

}
else 将学生 i 从队列摘除 ;
else
if 学生 i 的标识含 A then 将学生 i 从队列摘除 ;

else 取消学生 i 的 Sn 标识 ;
V ( mutex ) ;
if ( K2 < 10 ) & ( c1 < 100 ) then goto L2 ; }

}

coend .

```

51 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
VAR use : array [1 , 10000] of Boolean ; / * 该帐号是否被锁住使用标志

S : array [ 1 , 10000 ] of condition ; / * 条件变量

DEFINE lockaccount unlockaccount / * 移出过程

USE wait , signal , check , release ; / * 移入过程
procedure lockaccount ( var i,j : integer )

Begin
Check ( IM )
if i > j then begin

Temp:= i ;

i := j ;

j := temp ;

end ; / * 层次分配，先占号码小的账号否则可能产生死锁

if use [i] then wait(s[i].lockaccount,IM ) ;

else use [ i ] :=true ; / * 锁住 account ( i )

```



```

if use[j] then wait ( s[j].lockaccount , IM ) ;

else use [j] :=true ; / * 锁住 accounto )

```

```

Release ( IM ) ;
end ;

```

```

Proeedure unfockaccount ( var i:sinteger ; )

```

```

Begin

```

```

Check ( IM ) ;

```

```

use [ i ] := sfalse ;

```

```

signal(s[i].lock-account , IM ) ;

```

```

Release ( IM ) ;

```

```

end

```

```

begin

```

```

for i:= 1 ; to 10000 do use [i]:=false ;

```

```

end .

```

```

main ( )

```

```

{

```

```

cobegin

```

```

Process transfer account

```

```

begin

```

```

input a information line ;

```

```

get the account number i,j and 还款数 x ;

```

```

Lock-account.slockaccount ( 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-ccount.unlockaccount(i);

```

```

Lock-account.unlockaccount(j);

```

```

end ;

```

```

CoeDd .

```

52、某高校开设网络课程并安排上机实习，如果机房共有  $2m$  台机器，有  $2n$  个学生选课，规定：（1）每两个学生分成一组，并占用一台机器，协同完成上机实习；（2）仅当一组两个学生到齐，并且机房机器有空闲时，该组学生才能进机房；（3）上机实习由一名教师检查，检查完毕，一组学生同时离开机房。试用信号量和 P、V 操作模拟上机实习过程。

答：

```
var mutex , enter:semaphore ;
```

```
mutex := 1 ; enter := 0 ;
```

```
finish:=test:=rc:=0;computercounter:=2m;
```

```
cobegin
```

```
{
```

```
process studenti ( i=1 , 2 , , )
```

```
begin
```

```
P ( computereounter ) ; /
```

\* 申请计算机

```
P ( mutex ) ;
```

```
rc := rc + 1 ; /
```

\* 学生互斥计数

```
if rc == 1 then { v ( mutex ) ; P ( enter ) ; } /
```

\* 若只来一个学生，

```
则在即 ter 上等待
```

```
else { rc:= 0 ; V ( mutex ) ; V ( enter ) ; } s/
```

\* 到达一组中第二

```
个学生， rc 清。是为下一组计数用学生进入机房，上机实习；
```

```
V ( finish ) ; /
```

\* 告诉老师，实习

```
结束
```

```
P ( test ) ; /
```

\* 等待老师检查实习

```
结果
```

```
V( computercounter ) ; /
```

\* 归还计算机

```
end
```

```
process teacher
```

```
begin
```

```
P ( finish ) ; /
```

\* 等第一个学生实习

```
结束
```

```
P ( finish ) ; /
```

\* 等第二个学生实习

```
结束
```

```
检查实习结果；
```

```
V ( test ) ; /
```

\* 第一个学生检查完成

```
V ( test ) ; /
```

\* 第二个学生检查完成

```
end
```

```
}
```

```
coend .
```

53 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。若每次入、取水仅为 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。

```
var mutex1, mutex2 : semaphore ;
empty, full : semaphore ;
count : integer ;
mutex1 : mutex2 := 1 ; count := 3 ; empty := 10 ; full := 0 ;
cobegin
{
process 小和尚 (打水) i (i = 1, 2, ..., 3)
begin
repeat
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
until false ;
end

process 老和尚 (取水) j (j = 1, 2, ..., 3)
begin
repeat
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
until false ;
end
}
coend .
```

54 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。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
VAR free : array [ 0 , n - 1 ] of Boolean ;
waitcondition : condition ; i : integer ;
DEFINE acquire , release ;
USE check , wait , signal , return ;
```

```
procedure acquire ( var index : integer ; )
begin
  check ( IM ) ;
  for i := 0 to n - 1 do
    if free[i] then { free [i] := false ; index := i ; }
  else wait ( waitcondition , IM ) ;
  release ( IM ) ;
end
```

```
procedure return ( var index : integer ; )
begin
  check ( IM ) ;
  free[index]:=true ;
  signal ( waitcondition , IM ) ;
  release ( IM ) ;
end
begin
  for index := 0 to n - 1 do free[index]:=true ;
end
```

进程调用管程申请和归还页框的过程从略。

55、AND 型信号量机制是记录型信号量的扩充，在 P 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  $SP ( S_1, S_2, \dots, S_n )$  和  $VS ( S_1, S_2, \dots, S_n )$  其定义为如下的原语操作：

```
procedure SP ( vars , , , sn : semaphore )
begin
  if  $S_1 \geq 1$  & , &  $S_n \geq 1$  then begin
    for i := 1 to n do
       $S_i := S_i - 1$  ;
    end
  else begin
```

```
    { 进程进入第一个遇到的满足  $S_i < 1$  条件的  $S_1$  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。 } ;
```

end

procedure VP ( var S1 , , , Sn:semaphore ) begin

for i := 1 to n do begin

Si := S1 + 1 ;

{ 从所有 s 。信号量等待队列中移出进程并置入就绪队列。 } ;

end

试回答 AND信号量机制的主要特点，适用于什么场合？

S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：把进程在整个运行期间所要的临界资源，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。

56、试用 AND型信号量和 SP、SV操作解决生产者—消费者问题。

答：

Var B : array [ 0 , , k -1 ] of item ;

sput : semaphore := k ; / \* 指示有可用的空缓冲区的信号量

sget : semaphore := 0 ; / \* 指示缓冲区有可用的产品信号量

mutex : semaphore := 1 ; / \* 互斥信号量

sput := k ; / \* 缓冲区允许放入的产品数

sget := 0 ; / \* 缓冲区内没有产品

in : integer := 0 ;

out : Integer := 0 ;

begin

cobegin

process producer\_i

begin

L1 : produce a product ;

SP ( sput , mutex ) ;

B [ in ]:= product ;

in := ( in + 1 ) mod k ;

SV ( mutex , sget ) ;

goto L1 ;

end ;

```

process consumer_j
begin
L2 : SP ( sget , mutex ) ;
Product := B[out] ;
out := [out + 1] mod k ;
SV ( mutex , sput ) ;
consume a product :
goto L2 ;
end ;
coend
end

```

57、 试用 AND型信号量和 SP、SV 操作解决五个哲学家吃通心面问题。答：

```

Var fork[i] : array [ 0 , 4 ] of semaphore ;
fork[i] := 1 ;
cobegin
process Pi /* i = 0 , 1 , 2 , 3 */
begin
L1 :
思考 ;
SP ( fork [ i ] , fork [ i + 1 ] mod 5 ) ; /* 1 = 4 时 ,
SP ( fork [ 0 ] , fork [ 4 ] ) */

```

吃通心面；

V(fork[i],Vfork[i+1] mod 5);

Goto L1;

End;

58、 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 goto 语句，转向 if 语句再次检测每种资源状况。

59、一般型信号量机制（参见汤子派等编著的计算机操作系统，西安电子科技大学出版社）

对 AND型信号量机制作扩充，便形成了一般型信号量机制，SP ( s1;,t1 , d1, ; , ; sn , tn , dn ) 和 SV ( s1 ,d1 ; , sn,tn,dn ) 的定义如下：

```

procedure SP ( s1 , t1 , d1 ; , : sn , tn , dn )

```

```

var S1 , , , Sn : semaphore ;

```

```

t1 : , , tn : integer ;

```

```

dl , , , dn : integer ;

```

```

begin
if  $S1 \geq t1$  & , &  $S_n \geq T_n$  then begin
for  $i := 1$  to  $n$  do
 $S1 := S1 - d_i$ ;
end
else
{ 进程进入第一个遇到的满足  $s_i < t_i$  条件的  $S1$  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。 } ;

end
end

procedure SV (  $S1, d1$  ; ,  $s_n, d_n$  )
var  $S1$  , ,  $S_n$  : semaphore ;
 $d1$  , ,  $d_n$  : integer ;
begin
for  $i := 1$  to  $n$  do begin
 $S1 := S1 + d_i$  ;
{ 从所有  $s$  。信号量等待队列中移出进程并置入就绪队列。 } ;

end
end

```

其中， $t_i$  为这类临界资源的阈值， $d_i$  为这类临界资源的本次请求数。试回答一般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阈值  $t$ 。对 AND 型信号量机制作扩充，便形成了一般型信号量机制。

60 下面是一般信号量的一些特殊情况：

$SP(s, d, d)$

$SP(s, 1, 1)$

$SP(s, 1, 0)$

试解释它们的物理含义或所起的作用。

答：

$SP(s, d, d)$  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。

$sP(s, 1, 1)$  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)。

$sP(s, 1, 0)$  这是一个特殊且很有用的信号量, 当  $s \geq 1$  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。

61、试利用一般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  $m$  个读者同时读。为此, 又引入了一个信号量  $L$ , 赋予其初值为  $m$ , 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  $m$  个读者进入读后,  $L$  便减为 0, 而第  $m+1$  个读者必然会因执行  $sP(L, 1, 1)$  操作失败而被封锁。

利用一般信号量机制解决读者—写者问题的算法描述如下:

```
var m : integer ;           /          * 允许同时读的读进程数
L : semaphore := m ;       /          * 控制读进程数信号量, 最多 m
W : semaphore := 1 ;
begin
cobegin
process reader
begin
repeat
SP ( L , 1 , 1 ; W , 1 , 0 ) ;
Read the file ;
SV ( L , 1 ) ;
until false ;
end
process writer
begin
Repeat
SP ( W , 1 , 1 ; L , rn , 0 ) ;
Write the file ;
SV ( W , 1 ) ;
until false ;
end
coend
end .
```

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。  $sP(w, 1, 1; L, rn, 0)$  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。



