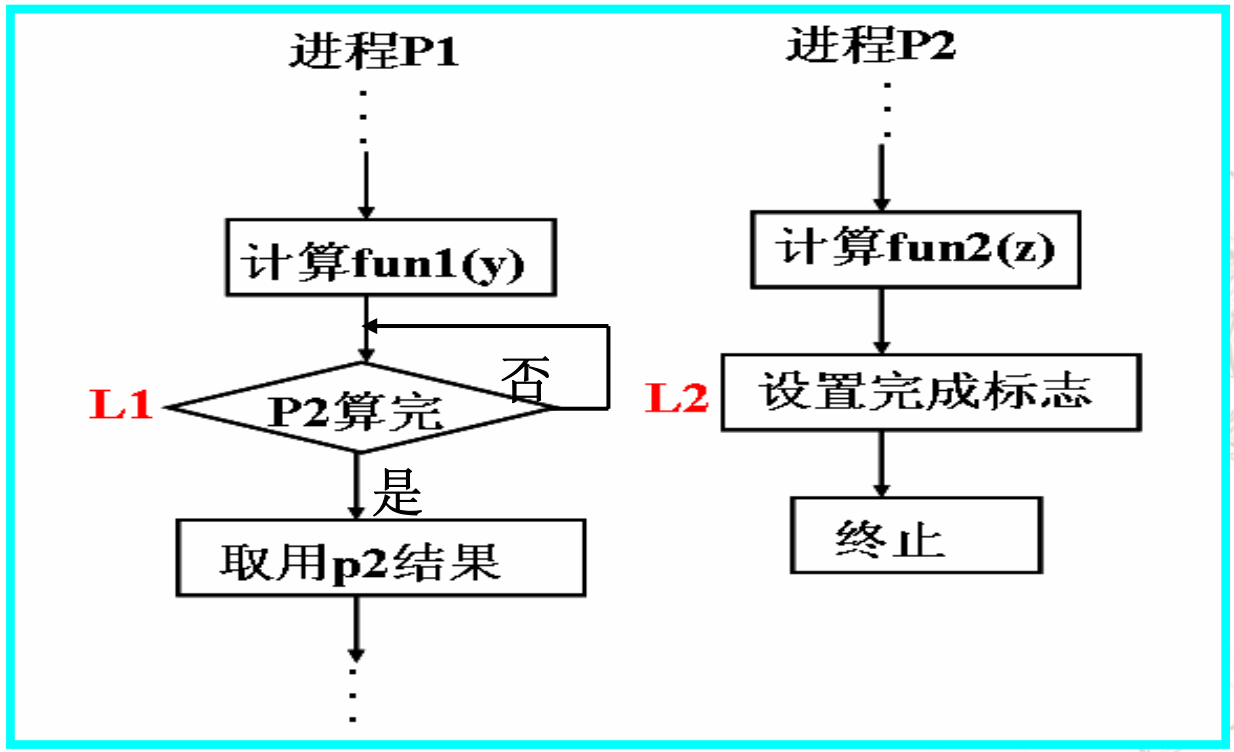
第四章 进程通信机制

——显式实现互斥、同步、通信关系的机制

§1 同步与互斥

一、进程合作

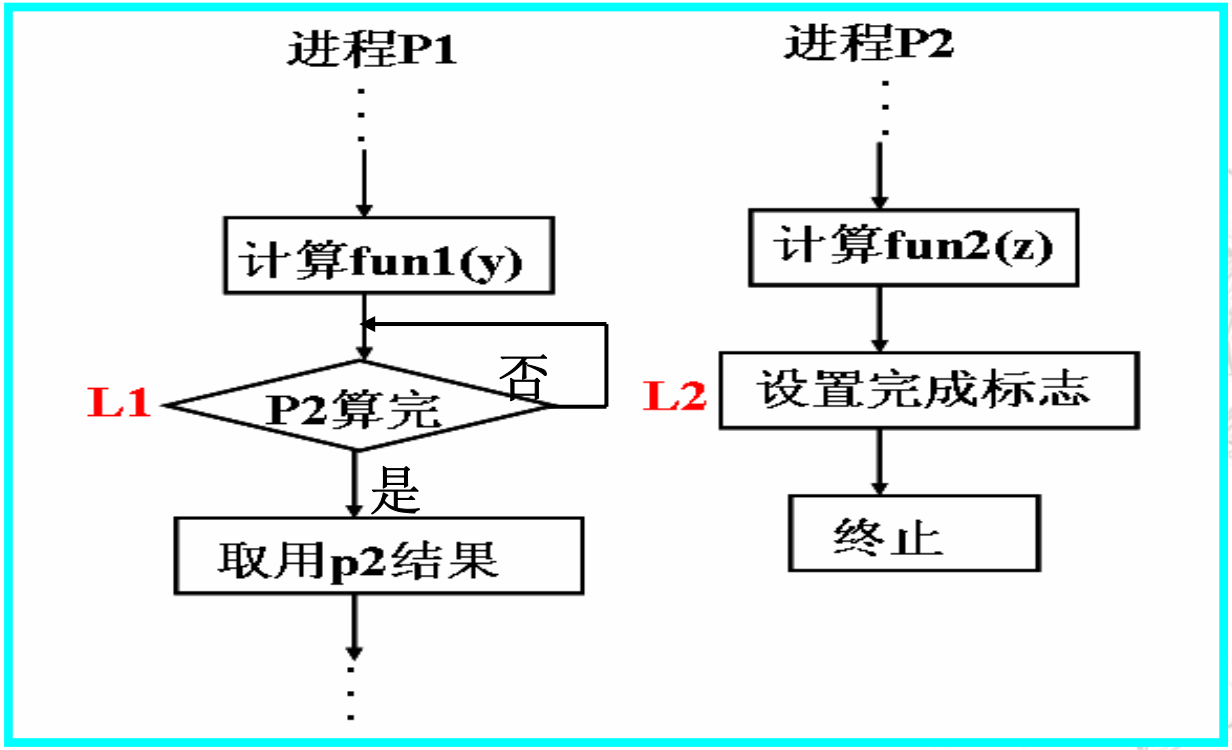
例:计算x=fun1(y)*fun2(z) 之值



§1 同步与互斥

一、进程合作

合作进程在执行次序上的协调关系称为同步(synchronization)关系。具体体现为一个进程到达某点后，在尚未得到伙伴进程发来的消息或信号时应阻塞自己，等待消息或信号到达后被唤醒继续执行。



§1 同步与互斥

二、共享资源

1.与时间有关的错误

例1:进程p1、p2共享一台打印机，编程控制对打印机的正确使用

与时间有关的错误：死锁

例2:飞机订票系统有两台终端，分别运行进程T1和T2。公共变量x代表订票数，则每台终端在订票后需对x进行加1操作

与时间有关的错误：结果不唯一

go

§1 同步与互斥

```
begin
  var x:integer:=1;  //x 为空闲打印机数
parbegin
   process p1
    begin
     repeat
       计算;
       x:=x-1;
       if x<0 then  block;
         使用打印机;
        x:=x+1;
     until false
    end

   process p2
      begin
       repeat
         计算;
         x:=x-1;
         if x<0 then  block;
           使用打印机;
          x:=x+1;
       until false
      end
parend
  end
```

§1 同步与互斥

```
begin
 var x:integer:=0;  //x 某航班飞机订票数
parbegin
   process T1
    begin
     repeat
       read(x);        r1:=x;
       x:=x+1;         r1:=r1+1;
       write(x);       x :=r1;
     until false
    end
   process T2
      begin
       repeat
         read(x);      r2:=x;
         x:=x+1;       r2:=r2+1;
         write(x);     x :=r2;
       until false
      end
parend
end
```

§1 同步与互斥

二、共享资源

1.与时间有关的错误

2. 临界资源CR (Critical Resource)

一次(一段时间内)只能由一个进程使用的资源

3. 临界区CS (Critical Section)

进程的程序文本中访问临界资源的代码段

4. 互斥(mutual exclusion)

进程异步并发，但不能同时而必须相互排斥的进入临界区

§1 同步与互斥

三、同步机构设计准则

```
repeat
 entry   section;
 critical section;
 exit   section;
  remainder section;
until false;
```

空闲让进

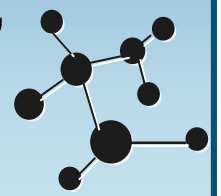
当没有进程处于临界区中，应允许一个请求者进程立即进入临界区，以有效利用CR

忙则等待

若有进程处于临界区，则其它试图进入临界区的进程必须等待，以保证CR的互斥使用

有限等待

进程等待进入临界区的时间应是有限的，而不应该陷入“死等”



让权等待

若进程不能进入临界区而等待时，应释放CPU资源，避免CPU“忙式等待”

§1 同步与互斥

例1：并发进程中涉及相同变量的程序段叫做 临界区 ，对这些程序段要 互斥 执行。

例2：在多进程系统中，为了保证公共变量的完整性，各进程应互斥地进入临界区。所谓临界区是指 D 。

A.一个缓冲区 B.一个数据区 C.同步机构 D.一段程序

例3：如果多个进程共享系统资源或相互合作完成一个共同的任务，则诸进程是以 G 方式运行的。对临界资源访问时采用 C 方式，对于相互合作的进程采用 D 方式以协调各进程执行的 E 。

A.共享 B.独立 C.互斥 D.同步 E.次序 F.次数 G.异步

例4：若系统中有五个并发进程涉及某个相同的变量A，则变量A的相关临界区是由 C 个临界区构成。

A.1 B.3 C.5 D.6

§1 同步与互斥

例5：有两个并发执行的进程P1和P2，共享初值为1的变量x。P1对x加1，P2对x减1。加1和减1操作的指令序列分别如下所示：

//加1操作	//减1操作
load R1,x //取x到寄存器R1中	load R2,x
inc R1	dec R2
store x,R1//将R1的内容存入x	store x,R2

两个操作完成后，x的值__C__。

A.可能为-1或3　　B.只能为1

C.可能为0、1或2　　D.可能为-1、0、1或2

§2 硬件同步机制

一、Test & Set 技术

1. 设置(set)

针对CR，设立一把锁Lock：

当Lock为假，表示CR空闲；

当Lock为真，表示CR忙碌

2. 关锁(test) —关锁原语

测试Lock之值：

若为真，不断测试等待；

若为假，则置Lock为真

3. 进入CS，使用CR

4. 开锁 —开锁原语

置Lock为假

二、TS指令

```
boolean TS(boolean lock)
 {  boolean old;
    old=lock; /*测试*/
    lock=true; /*关锁*/
    return  old; }
```

三、互斥模板

```
while(1)
{  while TS(lock); /*do skip */
   critical section;
   lock=false;
  remainder section;
}
```

§2 硬件同步机制

例：进程P1和P2共享打印机一台，试使用test&set技术编写它们正确并发的程序。

```
begin
 var x:integer:=1;  //x 为空闲打印机数
 parbegin
   process P1
    begin
     repeat
       计算；
       x:=x-1;
       if x<0 then  block;
         使用打印机；
        x:=x+1;
     until false
    end
   process P2
    begin
     repeat
       计算；
       x:=x-1;
       if x<0 then  block;
         使用打印机；
        x:=x+1;
     until false
    end
 parend
end
```

死锁！

§2 硬件同步机制

例：进程P1和P2共享打印机一台，试使用test&set技术编写它们正确并发的程序。

```
begin
 var x:Lock:=false;
parbegin
   process P1
    begin
     repeat
       计算；
      while TS(x);
        使用打印机；
      x:=false;
     until false
    end
```

```
   process P2
     begin
      repeat
       计算；
       while TS(x);
         使用打印机；
       x:=false;
      until false
     end
parend
end
```

§3 信号量机制

一、信号量含义

信号量(semaphore)是联系与控制CR的数据结构。其中，记录型信号量的定义如下：

```
typedef  struct
 { int value;
  /*信号量之值表示CR不同状态，可且仅可由P、V操作
   加以改变*/
  struct  process_control_bolck *list;
  /*list是因信号量而等待的进程队列指针*/
}semaphore;
```

§3 信号量机制

二、PV的数学定义

设S为信号量,则P(S) 意味着：

(1) S之值减1。

(2)测试S之值：

若S之值≥0,进程继续并发；

若S之值<0,进程阻塞并插入S的阻塞队列S.list中。

V(S)操作意味着：

(1)S值增1。

(2)测试：

若S之值>0,进程继续并发；

若S之值≤0,唤醒S.list队列的某一进程。

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

Test(Probera) →wait

```
V(semaphore S)
{
 S.value:=S.value+1;
 if S.value≤0  wakeup(S.list);
}
```

Increment(Verhogea) →signal

§3 信号量机制

三、PV的物理含义

- P操作意味着执行P操作的进程申请一个单位的CR
  V操作意味着执行V操作的进程释放一个单位的CR

- S之值的含义：
  —当s之值大于0时， s.value表示CR的可用数目
  —当s之值等于0时， s.value表示无空闲CR,亦无阻塞进程
  —当s之值小于0时， | s.value |表示因CR而阻塞的进程数

- P、V是同步原语，其执行不可分割。否则可能导致死锁发生

go

§3 信号量机制

例1：P1、P2共享打印机一台，试编写它们正确并发的程序。

```
begin
 var mutex:semaphore:=1;
 parbegin
    process P1
     begin
      repeat
        计算；
          P(mutex);
          使用打印机；
          V(mutex);
      until false
     end
```

```
    process P2
       begin
        repeat
          计算；
          P(mutex);
            使用打印机；
          V(mutex);
        until false
       end
 parend
end
```

§3 信号量机制

例2：A、B进程都要求互斥访问数据集D、E，编写它们正确并发的程序。

```
var Dm,Em:semaphore:=1,1;
begin
parbegin
 process A: begin
              P(Dm);
              P(Em);
              访问D、E;
              V(Dm);
               V(Em);
               其余部分;
               end
process B: begin
              P(Em);
              P(Dm);
              访问D、E;
              V(Em);
               V(Dm);
               其余部分;
               end
parend
end
```

死锁！

§3 信号量机制

四、AND型信号量

基本思想： 在一个原语中，将进入CS同时需要的多种临界资源，要么全部予以分配，要么一个都不分配。称为Swait(Simultaneous Wait),即同时wait操作或AND同步。

Swait(s_1,s_2,…s_n)

```
if s1≥1&s2≥1&…&sn≥1then
  for i=1 to n do
    si:=si-1;
  endfor
else
  place the process in the waiting queue associated with the
  first si found with si<1,and set the program count of this
  process to the beginning of swait operation
endif
```

§3 信号量机制

四、AND型信号量

基本思想： 在一个原语中，将进入CS同时需要的多种临界资源，要么全部予以分配，要么一个都不分配。称为Swait(Simultaneous Wait),即同时wait操作或AND同步。

Ssignal(s_1,s_2,…s_n)

for i=1 to n do

  s_i:=s_i+1;

  Remove all the process waiting in the queue associated with s_i into the ready queue.

endfor

endfor

§3 信号量机制

例：A、B进程都要求互斥访问数据集D、E，编写它们正确并发的程序。

```
var Dm,Em:semaphore:=1,1;
begin
parbegin
 process A: begin
              P(Dm);
              P(Em);        Swait(Dm,Em);
              访问D、E;
              V(Dm);
               V(Em);
               其余部分;
               end
process B: begin
              P(Em);
              P(Dm);
              访问D、E;
              V(Em);
               V(Dm);
               其余部分;
               end
parend
end
```

§3 信号量机制

五、一般信号量集

基本思想： 用于同时需要多种临界资源、每种所需数目不同、且可分配的资源还存在一个安全临界值时的处理。

Swait(s_1,t_1,d_1,…s_n,t_n,d_n)

if $s_1 \geqslant t_1$ & … & $s_n \geqslant t_n$ then
  for i=1 to n do
    $s_i := s_i - d_i$;
  endfor
else
  place the executing process in the waiting queue of the first s_i found with $s_i < t_i$, and set its program counter to the beginning of swait operation.
endif

§3 信号量机制

五、一般信号量集

基本思想： 用于同时需要多种临界资源、每种所需数目不同、且可分配的资源还存在一个安全临界值时的处理。

Ssignal(s_1,d_1,…s_n,d_n)

for i=1 to n do

s_i:=s_i+d_i;

Remove all the process waiting in the queue associated with s_i into the ready queue.

endfor

Swait(S, 1, 1)表示互斥信号量

Swait(S, 1, 0)作为可控开关

§4 信号量应用

一、实现互斥

```
var  mutex:semaphore:=1;    /*互斥信号量,初值为1*/
 repeat
  P(mutex);
  critical section;
  V(mutex);
  remainder section;
 until false;
```

互斥模板

§4 信号量应用

例1：设有10个进程共享同一互斥段，若最多允许3个进程进入互斥段，则使用的互斥信号量的初值是 A 。

A. 3　　B. 10　　C. 7　　D. 1

例2：若信号量S的初值为2，当前的值是-2，则表示有 2 个在该信号量上等待的进程。

例3：有20个进程共享一个互斥段，每次最多允许5个进程进入互斥段，则信号量的变化范围是 B 。

A. 0~5　　B. -15~5　　C. -19~1　　D. -1~5

例4:设与某资源相关联的信号量初值为3，当前值为1，若M表示该资源的可用数目，N表示等待该资源的进程数，则M、N分别是 B 。

A.0,1　　B.1,0　　C.1,2　　D.2,0

§4 信号量应用

例5：下面两个并发进程能正确运行吗？若不能，试举例说明并改正之。

```
var mutex:semaphore:=1;
begin
 parbegin
  var x:integer;
  procedure P1
   var y,z:integer;
    begin
   P(mutex);
   x:=1;
   y:=0;
   if x≥1 then y:=y+1;
   V(mutex);
   z:=y;
  end
  procedure P2
    var t,u:integer;
     begin
     P(mutex);
     x:=0;
     t:=0;
     if x<1 then t:=t+2;
     V(mutex);
     u:=t;
    end
 parend
end
```

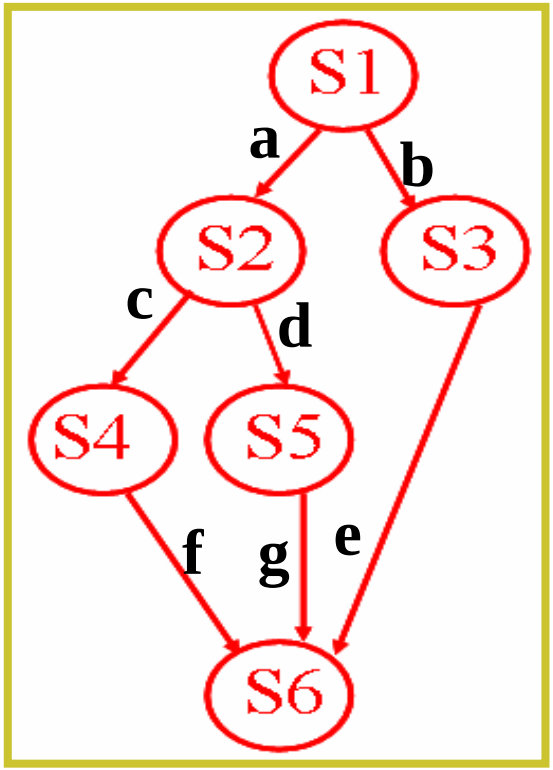
CS

CS

§4 信号量应用

二、描述前趋

- 针对一个偏序设置一个信号量,其初值为0。
- 前趋结点直接启动，后继结点需对信号量做p申请。
- 前趋结点完成，对信号量做v释放。




```
var a,b,c,d,e,f,g:semaphore:=0,0,0,0,0,0,0;
begin
parbegin
  begin S1; V(a); V(b); end;
  begin P(a); S2; V(c); V(d); end;
  begin P(b); S3; V(e); end;
  begin P(c); S4;V(f); end;
  begin P(d); S5; V(g); end;
  begin P(f); P(g); P(e); S6; end;
 parend
end
```

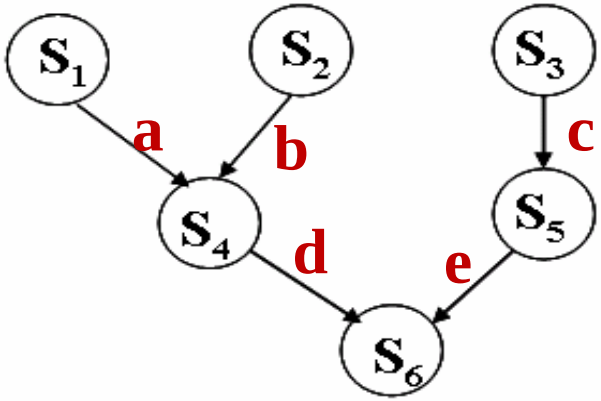
§4 信号量应用

例1:已知求值公式$(A^2+3B)/(B+5A)$,若A、B已赋值,试用P、V操作描述该公式的求解过程。

S1: x1=A*A　　S2: x2=3*B

S3: x3=5*A　　S4: x4=x1+x2

S5: x5=B+x3　　S6: x6=x4/x5




```
var a,b,c,d,e:semaphore;
a,b,c,d,e:=0,0,0,0,0;
begin
parbegin
  begin S1; V(a); end;
  begin  S2; V(b)end;
  begin S3; V(c); end;
  begin P(a); P(b);S4;V(d); end;
  begin P(c); S5; V(e); end;
  begin P(d); P(e); S6; end;
 parend
end
```

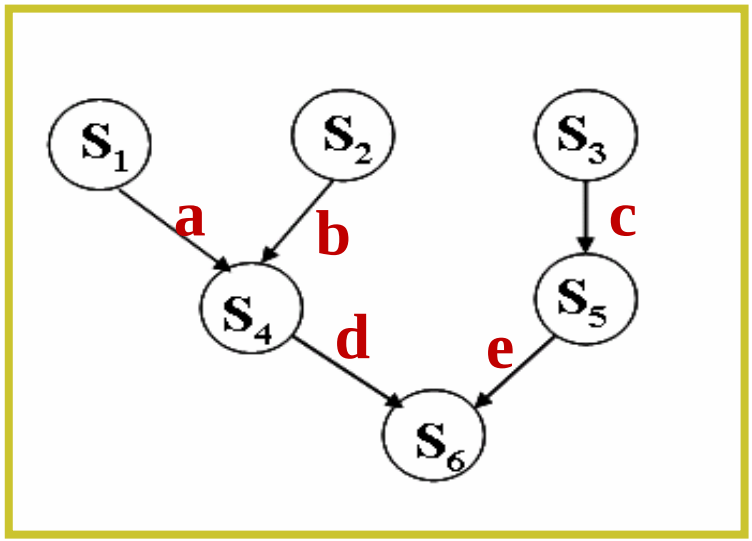
§4 信号量应用

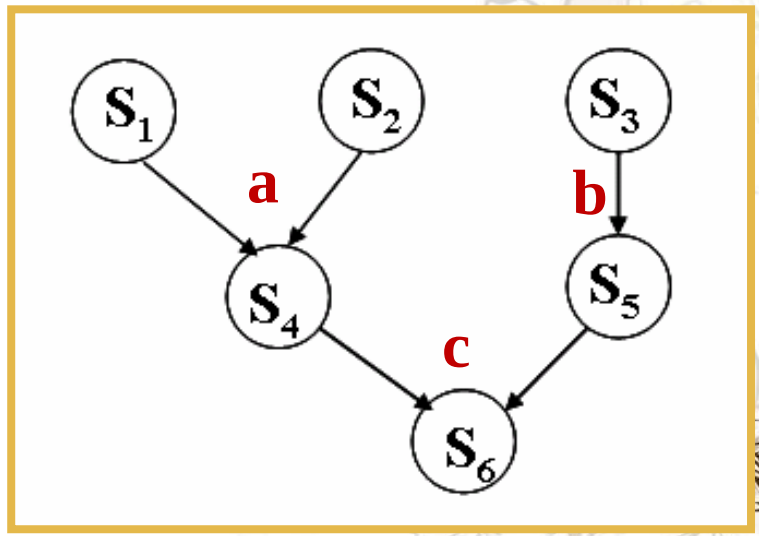
例1:已知求值公式$(A^2+3B)/(B+5A)$,若A、B已赋值,试用P、V操作描述该公式的求解过程。

S1: x1=A*A　　S2: x2=3*B

S3: x3=5*A　　S4: x4=x1+x2

S5: x5=B+x3　　S6: x6=x4/x5



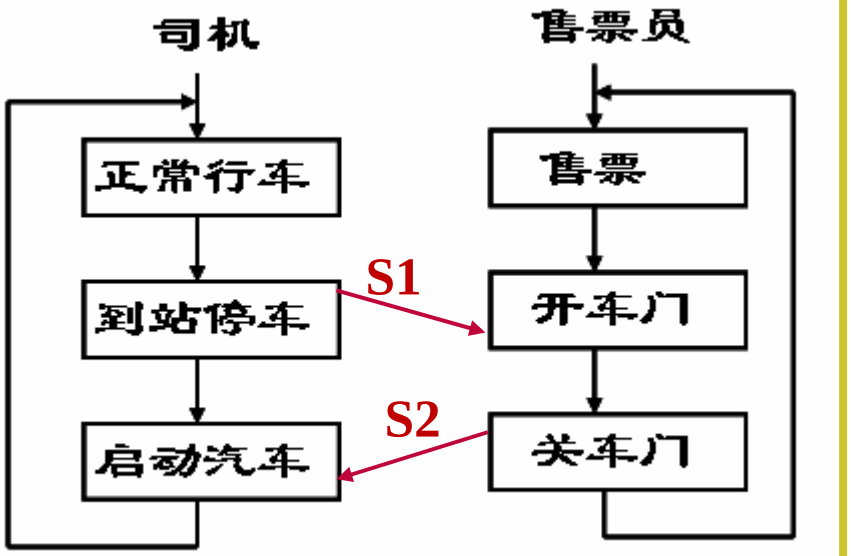


§4 信号量应用

例2 ：在一辆公共汽车上，司机负责驾驶车辆；售票员负责开关车门。二者的协调关系如下：

- 汽车到站驾驶员停车后，售票员才能打开车门上、下乘客；
- 售票员关好车门后，驾驶员才能再次启动汽车；

试写出司机和售票员正确并发的程序。



S1:能打开车门？初值为0

S2:能启动汽车？初值为0

§4 信号量应用

```
var s1,s2:semaphore:=0,0;
begin
 parbegin
  process 司机
   begin
    repeat
     行车;
     停车;
     V(s1);
     P(s2);
     启动汽车;
    until false
   end
```

```
process 售票员
   begin
    repeat
     售票;
     P(s1);
     开车门;
     关车门;
     V(s2);
    until false
   end
parend
end
```

§5 经典同步问题

一、生产者—消费者问题

问题描述： 一组生产者(producer)和一组消费者(consumer)通过一个容量为n的有界缓冲区(buffer)进行合作。生产者负责向缓冲区中投放产品(product)，而消费者负责从中取走产品进行消费。试写出生产者——消费者进程正确同步的程序。

empty—可供投放产品的空缓冲,初值n

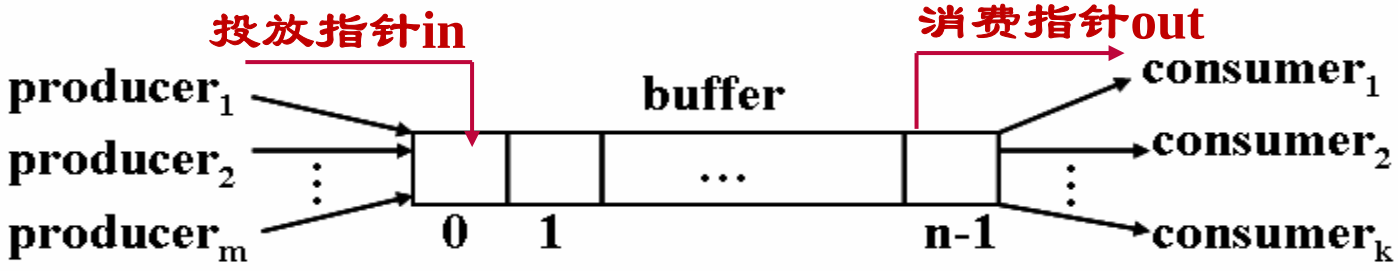
full—装有产品的满缓冲,初值0

mutex—buffer的互斥信号量,初值为1

读写指针in,out:0,1,2…n-1

in:=(in+1)mod n

out:=(out+1)mod n



§5 经典同步问题

一、生产者—消费者问题

```
var mutex,empty,full:semaphore:=1,n,0;
buffer:array[0..n-1] of product;
in,out:integer:=0,0;
begin
  parbegin
    producer: begin
                repeat
                  produce next product;
                  P(empty);
                  P(mutex);
                  buffer[in]:=product;
                  in:=(in+1)mod n;
                  V(full);
                  V(mutex);
                until false
              end
```

```
consumer:
 begin
    repeat
      P(full);
      P(mutex);
       m :=buffer[out];
       out:=(out+1)mod n;
       V(empty);
       V(mutex);
       consume product in m
     until false
 end
parend
end
```

§5 经典同步问题

例1 ：围棋工人将等量黑子和白子混装在一个箱子里，现用自动分拣系统将黑白子分开。该系统由两个并发执行的进程组成，系统功能如下：

◆process A专门拣黑子，process B专门拣白子；

◆每个进程每次只拣一个子，当一个进程拣子时，不允许另一个进程去拣子；

◆规定当一个进程拣完一个子，必须让另一个进程拣一个子；

试编写进程A、B正确并发的程序。

S1：进程A可拣黑子的权利，初值为1

S2：进程B可拣白子的权利，初值为0

§5 经典同步问题

```
var S1,S2:semaphore:=1,0;
begin
  parbegin
    Process A: begin
                 repeat
                  P(S1);
                   拣一个黑子;
                  V(S2);
                 until false
               end
    Process B: begin
                 repeat
                  P(S2);
                   拣一个白子;
                  V(S1);
                 until false
               end
  parend
end
```

§5 经典同步问题

例2：一个空盒可以存放一个水果，爸爸向内放苹果，妈妈向内放桔子；儿子专等吃盒中的桔子，女儿专等吃盒中的苹果。一次只能放或取一只水果。试问：

◆应设置几个信号量，分别是何含义？各取什么初值？

◆父亲和母亲间应实现 互斥 关系；

父亲和女儿应实现 同步 关系；

母亲和儿子应实现 同步 关系。

信号量设置：

s1—水果盘有空吗？初值为1

s2—盘中有苹果吗？初值为0

s3—盘中有桔子吗？初值为0

§5 经典同步问题

例3：三个进程P1、P2、P3互斥使用一个包含N(N>0)个单元的缓冲区。P1 每次用produce( )生成一个正整数并用put( )送入缓冲区某一空单元中；P2每次用getodd( )从该缓冲区中取出一个奇数并用countodd( )统计奇数个数； P3每次用geteven( )从该缓冲区中取出一个偶数并用counteven( )统计偶数个数。请用信号量机制实现三个进程的同步与互斥活动，并说明所定义的信号量的含义。要求用伪代码描述。

※同步信号量※

empty—空缓冲数目，初值为N。

full1—奇数产品个数，初值为0。

full2—偶数产品个数，初值为0。

※互斥信号量※

mutex—缓冲区互斥操作权，初值为1。

§5 经典同步问题

例3：三个进程P1、P2、P3互斥使用一个包含N(N>0)个单元的缓冲区。P1 每次用produce( )生成一个正整数并用put( )送入缓冲区某一空单元中；P2每次用getodd( )从该缓冲区中取出一个奇数并用countodd( )统计奇数个数； P3每次用geteven( )从该缓冲区中取出一个偶数并用counteven( )统计偶数个数。请用信号量机制实现这三个进程的同步与互斥活动。

P1	P2	P3
:	:	:
x=produce( );	P(full1);	P(full2);
P(empty);	P(mutex);	P(mutex);
P(mutex);	getodd( );	geteven( );
put( );	V(mutex);	V(mutex);
if(x%2≠0) V(full1);	V(empty);	V(empty);
else V(full2);	countodd( );	counteven( );
V(mutex);	:	:
:		

§5 经典同步问题

二、读者—写者问题

问题描述： 有一个共享文件L，

允许n个进程同时读L中信息；——读者

但任何时刻最多只能有一个进程写或修改L中信息；——写者

当有进程读时不允许任何进程写，当有进程写时不允许其它进程读或写；——读/写互斥、写/写互斥

写出读者和写者正确并发的程序。

分析

- 设readcount表示某时刻读者数,初值为0
- 定义信号量rmutex,实现对readcount的互斥访问
- 定义信号量wmutex,实现写/写互斥、读/写互斥

§5 经典同步问题

```
var rmutex,wmutex:semaphore:=1,1;
  readcount:integer:=0;
  begin
  parbegin
  Writer: begin
    repeat
    P(wmutex);
     perform write operation;
    V(wmutex);
   until false
  end
  Reader: begin
    repeat
    P(rmutex);
     if readcount=0 then P(wmutex);
     readcount:=readcount+1;
    V(rmutex);
      perform read operation;
    P(rmutex);
     readcount:=readcount-1;
     if readcount=0 then V(wmutex);
    V(rmutex);
   until false
  end
  parend
  end
```

go

§5 经典同步问题

例1：一个主修动物行为学、辅修计算机科学的学生参加了一个课题，调查花果山的猴子能否被教会理解死锁。他找到一处峡谷，横跨峡谷拉了一个绳索（假定为南北方向），这样猴子可以攀着绳索越过峡谷。只要它们朝着相同方向，同一时刻可以有多个猴子通过。但是如果在相反的方向上同时有猴子通过则会发生死锁（这些猴子被卡在绳索中间，假设这些猴子无法在绳索上从另一只猴子身上翻过去）。如果一只猴子想越过峡谷，它必须看当前是否有别的猴子在逆向通过。请使用信号量写一个避免死锁的程序来解决问题。

count1:正向通过绳索的猴子数,初值为0
count2:反向通过绳索的猴子数,初值为0
mutex1:控制对count1互斥访问的信号量,初值为1
mutex2:控制对count2互斥访问的信号量,初值为1
s:封锁相反方向的互斥信号量,初值为1

§5 经典同步问题

```
var S,mutex1,mutex2:semaphore:=1,1,1;
var count1,count2:integer:=0,0;
begin
parbegin
 process 正向通过
  begin
   P(mutex1);
   if count1=0 then P(S); //封锁反向通过权利
   count1:=count1+1;
   上绳索；
   V(mutex1);
   从正向通过绳索；
   P(mutex1);
   count1:=count1-1;
   下绳索；
   if count1=0 then V(S); //开放反向通过权利
  V(mutex1);
end
```

```
process 反向通过
  begin
   P(mutex2);
   if count2=0 then P(S);
   count2:=count2+1;
   上绳索；
   V(mutex2);
   从反向通过绳索；
   P(mutex2);
   count2:=count2-1;
   下绳索；
   if count2=0 then V(S);
  V(mutex2);
end
parend
end
```

§5 经典同步问题

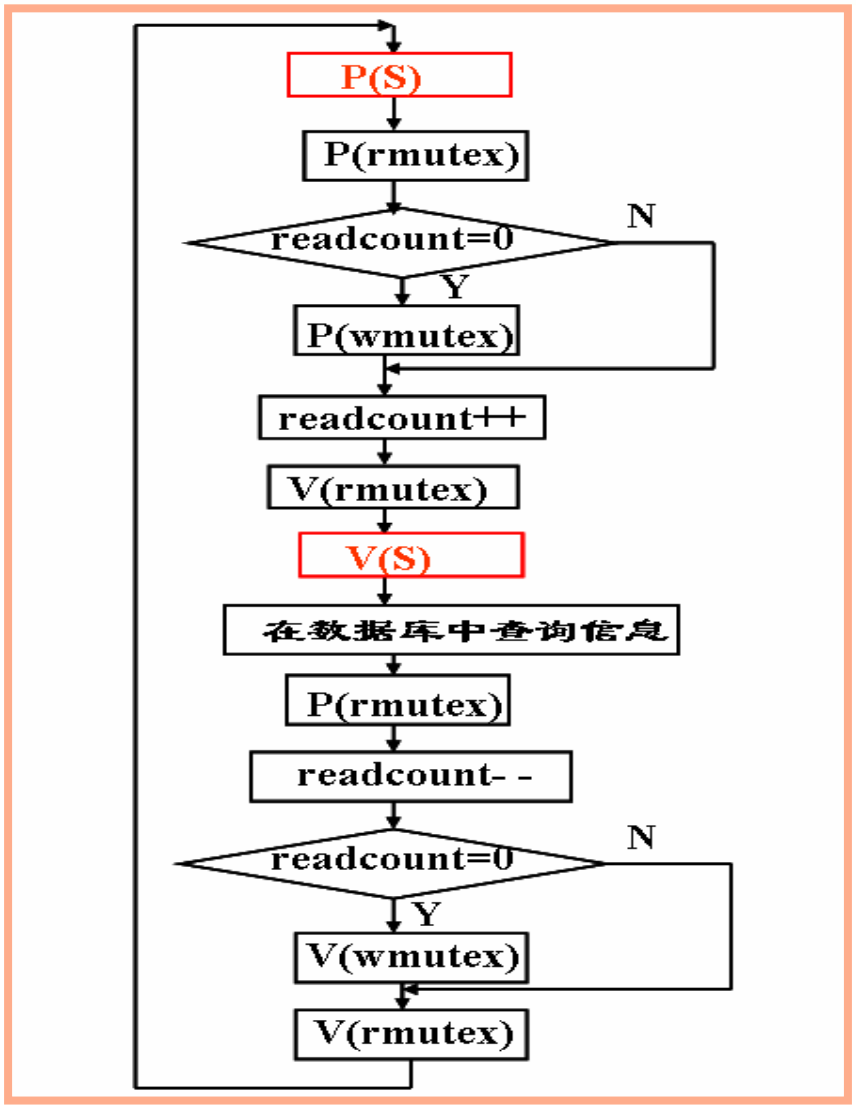
例2：在一个飞机订票系统中，多个用户共享一个数据库。多用户同时查询是可以接收的，假若一个用户要订票需更新数据库时，其余所有用户都不可以访问数据库。请画出用户查询和订票的逻辑框图。要求：当一个用户订票需要更新数据库时，不能因为不断有查询者的到来而使他长期等待。

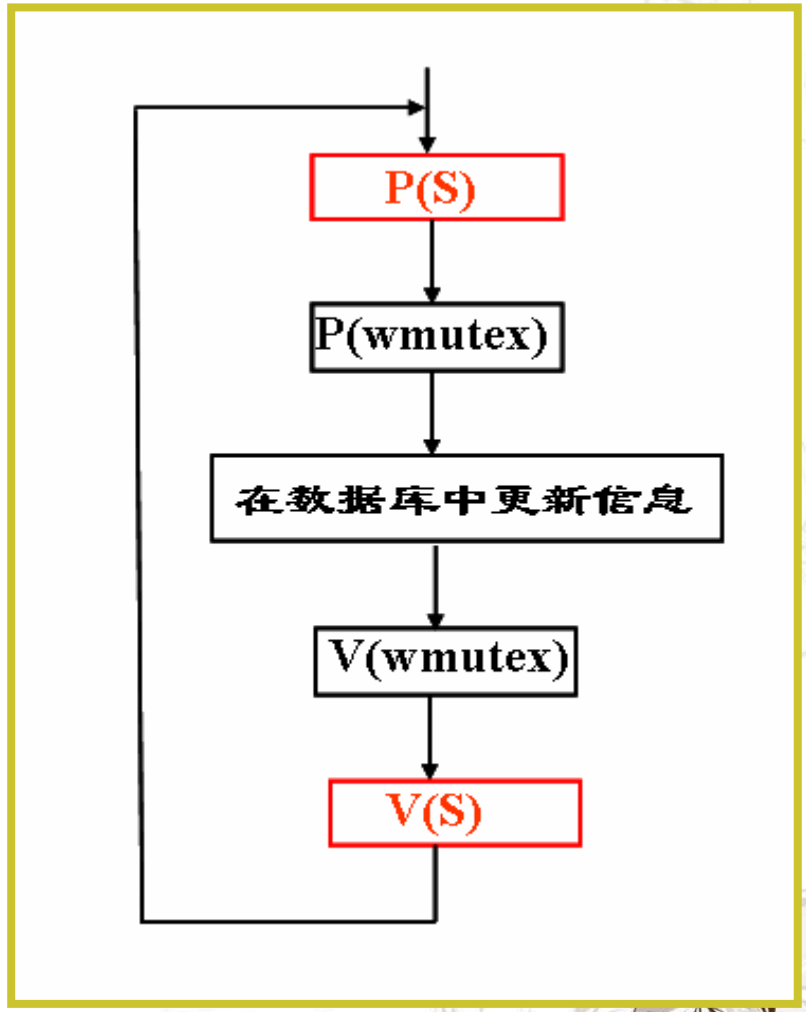
读者与写者问题：其中查询操作是读者，订票操作是写者

读写问题模板体现的是读者优先，本题要求写者优先

需增设一个信号量S体现写着优先信号，即S能让写者进程阻塞后继读者进程

§5 经典同步问题


P(S)
P(rmutex)
readcount=0
N
Y
P(wmutex)
readcount++
V(rmutex)
V(S)
在数据库中查询信息
P(rmutex)
readcount- -
readcount=0
N
Y
V(wmutex)
V(rmutex)

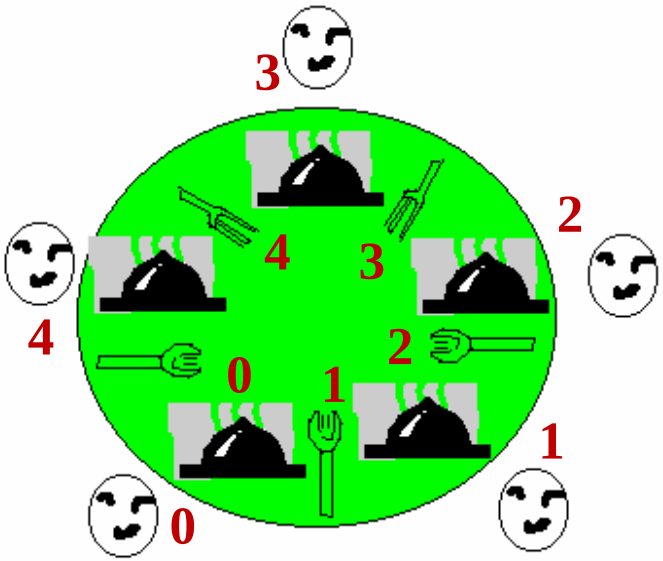

P(S)
P(wmutex)
在数据库中更新信息
V(wmutex)
V(S)

§5 经典同步问题

三、哲学家就餐问题

问题描述

5个哲学家围坐在一圆桌旁，每人面前一盘通心粉，每两人之间放一把叉子。哲学家的行为是思考和感到饥饿去吃通心粉。吃通心粉需同时使用两把叉子，规定只能取左右两边的叉子。写出控制哲学家正确并发的程序。



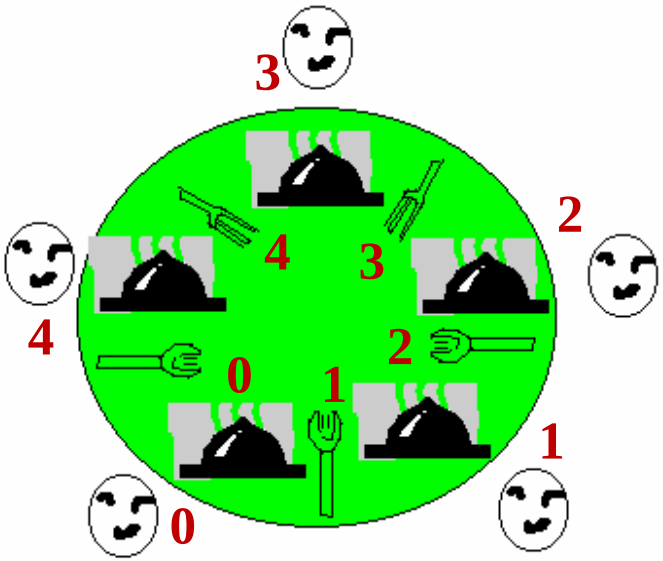

```
Process philosopher_i( )
{ while(1)
  { think;

    eat;

  }
}
```

§5 经典同步问题

三、哲学家就餐问题

问题描述

5个哲学家围坐在一圆桌旁，每人面前一盘通心粉，每两人之间放一把叉子。哲学家的行为是思考和感到饥饿去吃通心粉。吃通心粉需同时使用两把叉子，规定只能取左右两边的叉子。写出控制哲学家正确并发的程序。




```
semaphore fork[5] ={1,1,1,1,1};
Process  philosopher_i( )
{ while(1)
   {  think;
      P(fork[i]); P(fork[(i+1)%5]);
      eat;
      V(fork[i]);V(fork[(i+1)%5]);
   }
}
```

死锁

§5 经典同步问题

三、哲学家就餐问题

问题描述

5个哲学家围坐在一圆桌旁，每人面前一盘通心粉，每两人之间放一把叉子。哲学家的行为是思考和感到饥饿去吃通心粉。吃通心粉需同时使用两把叉子，规定只能取左右两边的叉子。写出控制哲学家正确并发的程序。

```
semaphore sm=4;
semaphore fork[5] ={1,1,1,1,1};
Process  philosopher_i( )
{ while(1)
   { think;
     P(sm);
     P(fork[i]); P(fork[(i+1)%5]);
      eat;
    V(fork[i]);V(fork[(i+1)%5]);
     V(sm);
    }}
```

```
semaphore fork[5] ={1,1,1,1,1};
Process  philosopher_i( )
{ while(1)
   {  think;
     P(fork[i]); P(fork[(i+1)%5]);
      eat;
     V(fork[i]);V(fork[(i+1)%5]);
    }
 }
```

死锁

§5 经典同步问题

三、哲学家就餐问题

问题描述

5个哲学家围坐在一圆桌旁，每人面前一盘通心粉，每两人之间放一把叉子。哲学家的行为是思考和感到饥饿去吃通心粉。吃通心粉需同时使用两把叉子，规定只能取左右两边的叉子。写出控制哲学家正确并发的程序。

```
semaphore fork[5] ={1,1,1,1,1};
  Process philosopher_i( )
  { while(1)
    { think;
     swait(fork[i],fork[(i+1)%5]);
       eat;
    ssignal(fork[i],fork[(i+1)%5]);
     }
  }
```

```
semaphore fork[5] ={1,1,1,1,1};
Process  philosopher_i( )
{ while(1)
   {  think;
     P(fork[i]); P(fork[(i+1)%5]);
      eat;
     V(fork[i]);V(fork[(i+1)%5]);
   }
}
```

死锁

§6 进程高级通信

一、低级通信和高级通信

共同点：均实现了进程间的信息交互

同步机构 — 低级通信

- **通信效率低**：同步机构只能传递少量控制信息或数据，传递数据量少
- **通信过程不透明 (可见)**：用户在程序中实现通信的细节，编程复杂，易出错

区别

进程通信 — 高级通信

- **高效性**：可借助通信命令一次性的在进程间传送大量数据
- **透明性(不可见)**：通信具体细节由OS实现并隐藏。简化了通信程序的设计

§6 进程高级通信

二、单机系统高级通信机制

共享存储区通信

- 通信进程在内存划分出一个共享存储区域,诸进程通过对该区域的读写,实现通信
- 通信效率高

管道pipe通信

- 通信进程在辅存建立文件作为通信载体,发送者以写方式、接收者以读方式打开该文件,实现通信
- 通信成本低

消息传递系统

- 通信进程间以格式化的消息(message)作为通信单位，在网络中也称为报文
- 使用最为广泛

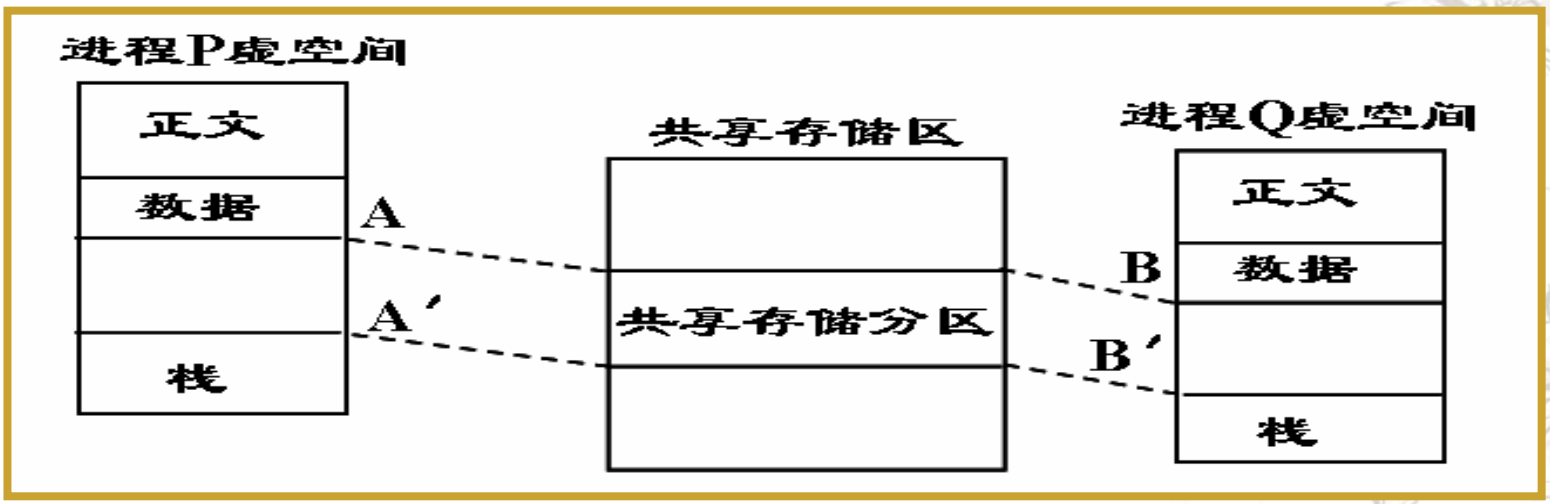
§6 进程高级通信

三、UNIX System V进程通信系统设计

§6 进程高级通信

四、共享存储区通信

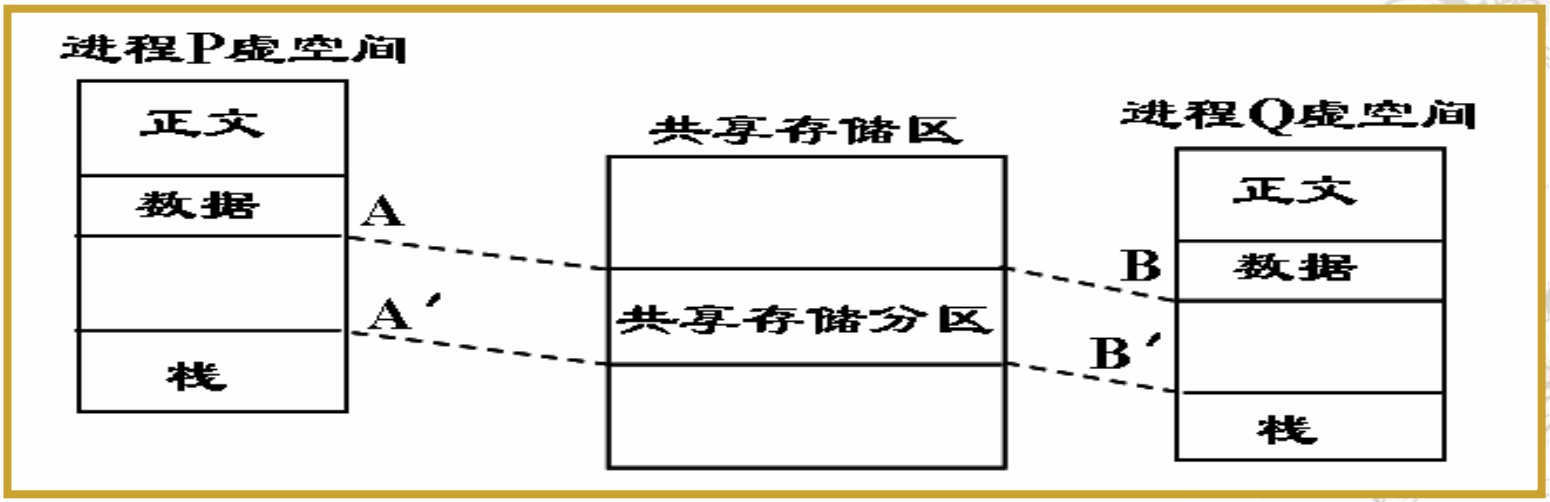
- 在主存设置shm区域(shared memory)作为进程间通信区域
- 建立分区(shmget):建立指定名称新分区或返回已存在分区的描述符shmid。 int shmget(key,size,flag);
- 附接分区(shmat):将共享分区附接到进程虚空间指定位置。 char *shmat(shmid,addr,flag);
- 读写虚空间： 由主存地址变换机构映射到指定共享分区。
- 断接(shmdt):拆除共享存储区与进程地址空间的连接。



§6 进程高级通信

四、共享存储区通信

- 通过对主存的访问实现通信，通信速度快、效率高。
- 一个共享存储区可以附接到多个进程上，一个进程的虚空间可附接多个不同的共享分区，通信灵活。
- 需要在主存维持共享存储区以供通信，通信开销大。



§6 进程高级通信

五、管道(pipe)通信

- 管道(pipe):连接读写进程，实现进程通信的共享文件。

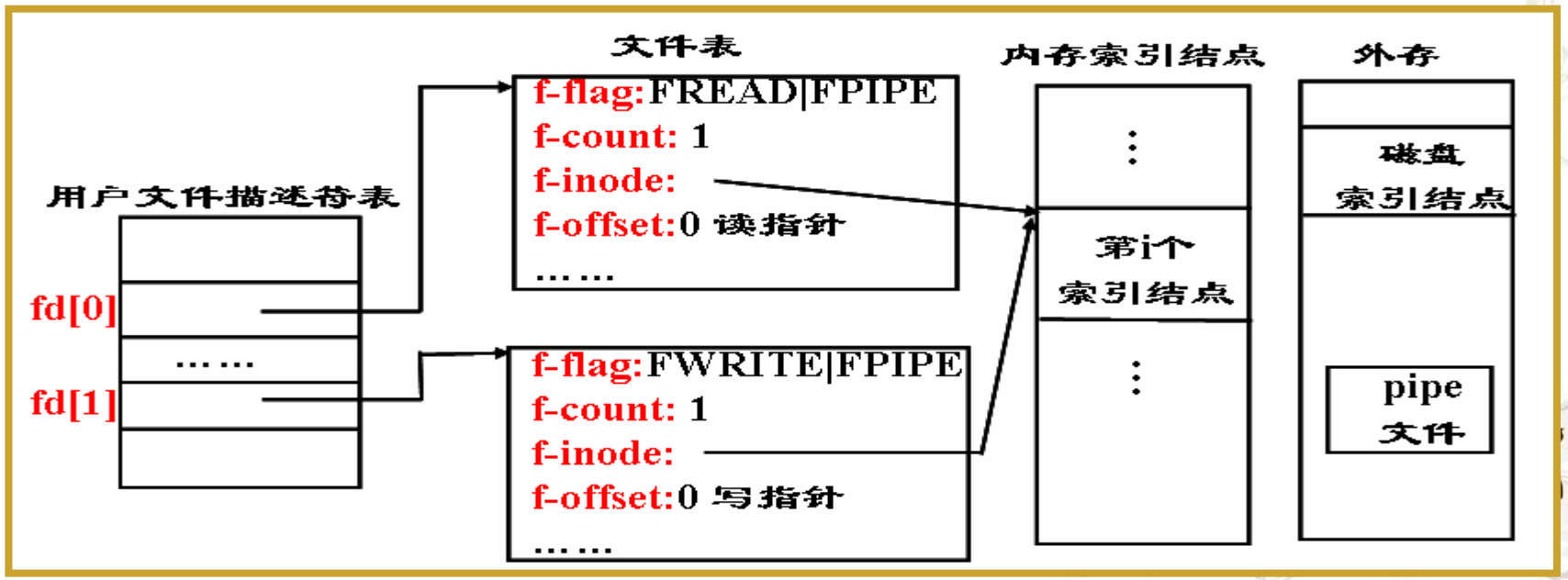
——无名管道：使用pipe建立的，无路径名、只用文件描述符加以识别的临时文件。适用于父子进程间或父进程安排的各个子进程之间进行通信。

——有名管道：使用mknod建立的，可以在文件系统中长期存在、具有路径名的文件。能感知其存在的进程均可使用路径名访问该文件。

§6 进程高级通信

五、管道(pipe)通信

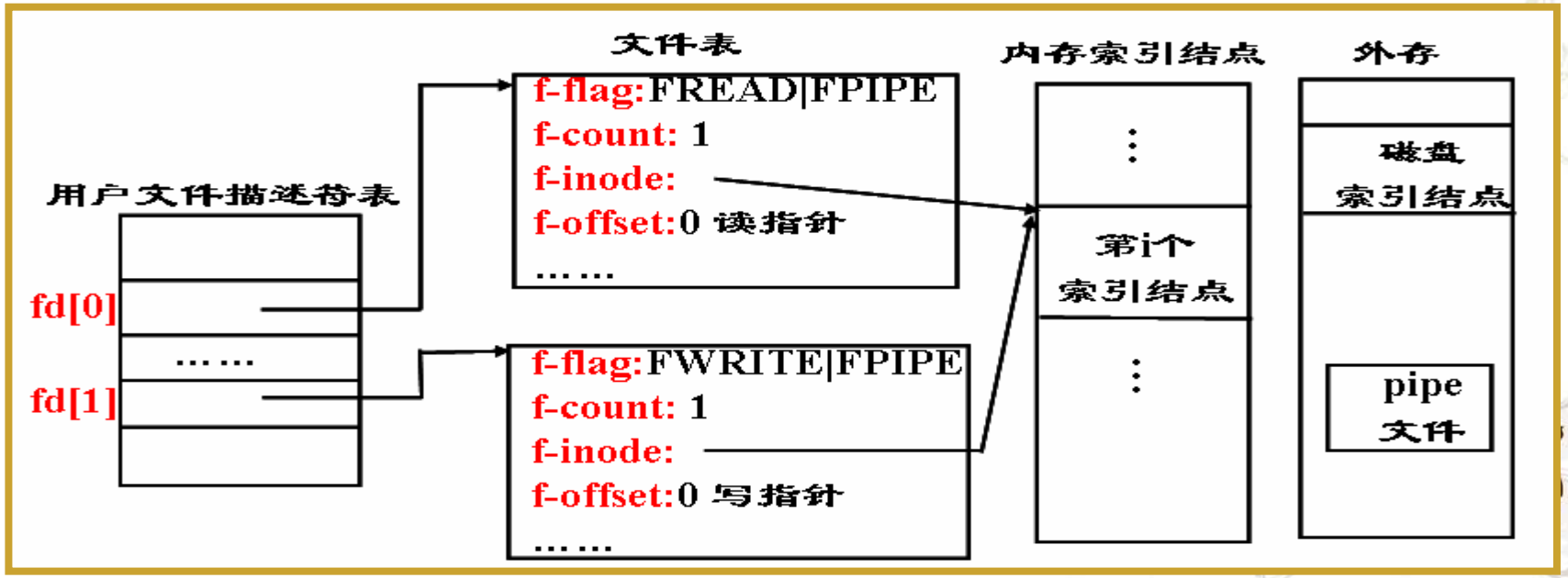
- 通过pipe系统调用创建无名管道:int pipe(int fd[2]) 。
  - —为pipe建立磁盘索引结点di_node和内存索引结点inode,
    若无空闲结点,失败;
  - —为读和写建立文件表项,若无空闲表项,失败;
  - —在用户文件描述符表中分配两个用户文件描述符。



§6 进程高级通信

五、管道(pipe)通信

- 通过pipe系统调用创建无名管道:int pipe(int fd[2]) 。
- fork子进程，父子进程共享pipe。



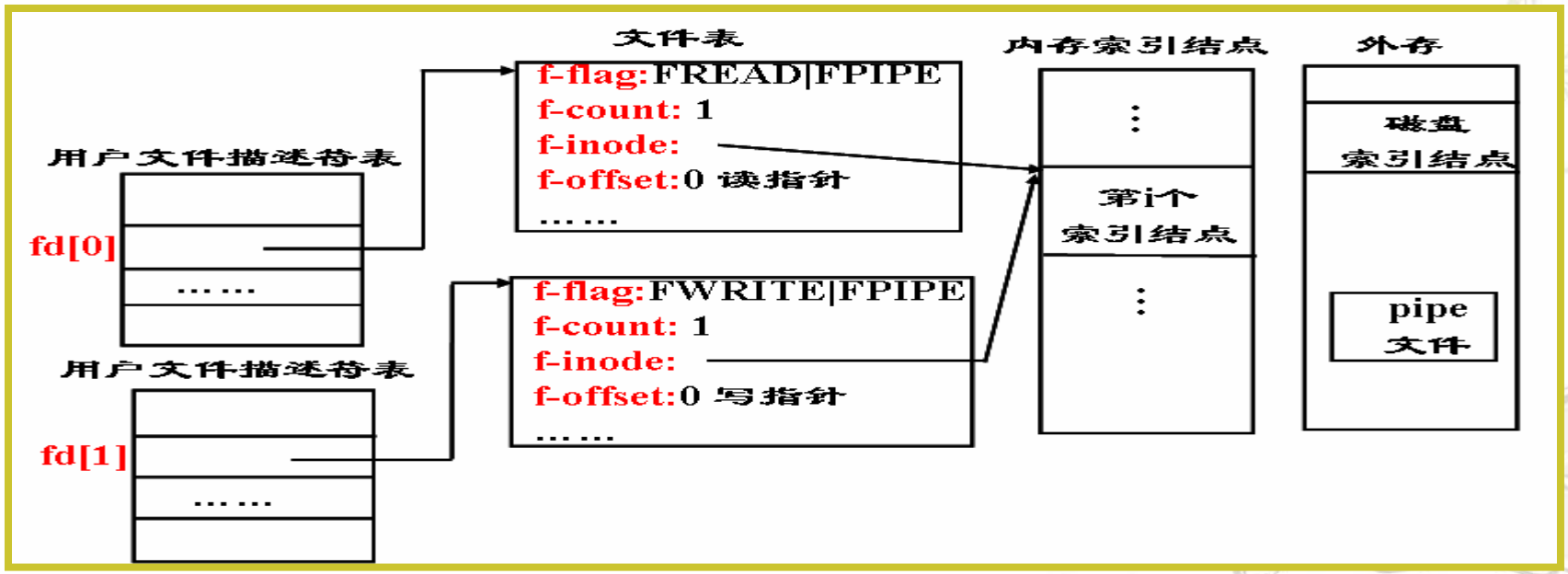
§6 进程高级通信

五、管道(pipe)通信

- 通过pipe系统调用创建无名管道:int pipe(int fd[2]) 。
- fork子进程，父子进程共享pipe。
- 通过系统调用write和read进行管道的读写。

  read(fd[0],主存首址,字节数);

  write(fd[1],主存首址,字节数);



§6 进程高级通信

五、管道(pipe)通信

- **通信协调:**

①通信双方**互斥**使用管道:通过对索引结点上锁实现。

②通信双方必须**知道**对方是否存在：若接收端已关闭而有等待写管道的进程，核心予以唤醒;若发送端已关闭而有等待读管道的进程，核心唤醒它们，这些进程从读调用中空手返回。

③**同步**关系：若无足够写空间则write阻塞，同样，当读进程读空管道时,要出现读阻塞，直到写进程将其唤醒。

- **pipe通信特点：**

① 以文件为载体，通信开销小。

② I/O次数多，通信速度较慢。

③ 通信能力较弱：无名管道仅限于进程家族内通信。

§6 进程高级通信

六、消息传递系统

- 共享存储区通信(shared memory:shm) —— 高速性但开销大
  —允许进程通过共享虚地址空间的部分区域实现通信
- 管道(pipe)通信 ——开销低但局限于进程家族内、速度较慢
  —允许进程以自然字符流方式借助文件传递数据
- 消息传递系统 —— 广泛性
  —允许在进程间传递消息(格式化数据)

※直接通信：发送者和接收者均显式指定对方标识符。

send(Receiver,message);receive(Sender,message);

※间接通信：以信箱为中间实体，发送者将消息发送至信箱，由核准的目标用户从信箱中读取。

send(mailbox,message);receive(mailbox,message);

§6 进程高级通信

七、消息缓冲队列通信

基于消息直接通信的实例

- **通信载体：** 在OS空间设置由若干缓冲区构成的通信缓冲池(瓶颈资源)，一个缓冲区可以收容一个格式化的消息。
- **PCB设置：** 在PCB中设置与消息缓冲通信有关的数据项。

```
type messagebuffer=record
sender; //发送者进程标识符
size;  //消息长度
text;  //消息正文
next;  //消息队列指针
end;
```

```
type processcontrolblock=record
  .
  .
 mg;  //消息队列队首指针
 mutex; //消息队列互斥信号量
 sm; //消息队列消息数
end;
```

§6 进程高级通信

七、消息缓冲队列通信

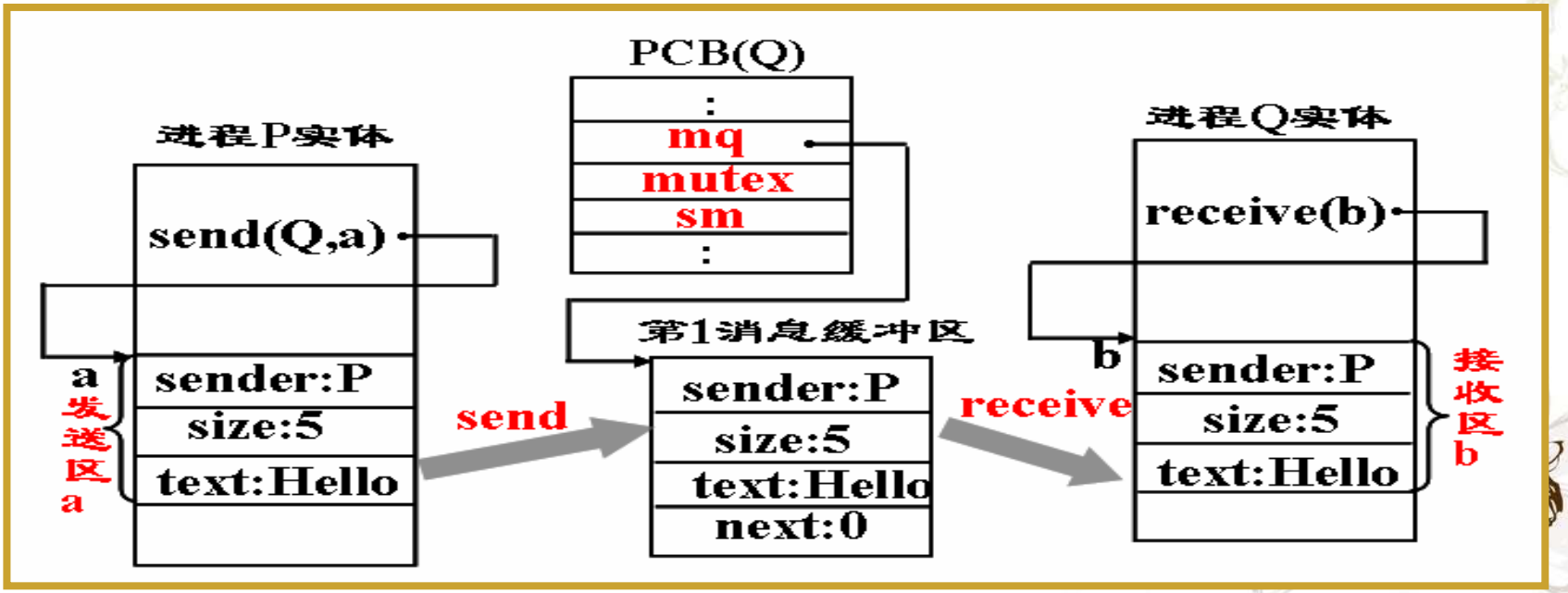
- 通信过程及协调：

(1)构成消息:发送进程在实体数据区内设置发送区,并填入消息

(2)发送消息:调用send原语形成消息缓冲区并挂至消息队列尾部

(3)接收消息:目标进程调用receive原语接收消息队列的队首消息

(4)同步与互斥:保证消息队列互斥操作并同步消息接收与发送



§6 进程高级通信

- 发送原语形式化描述

```
procedure send(receiver, a)
  begin
   getbuf(a.size, i); //获得通信缓冲池中的消息缓冲区i
   i.sender:=a.sender;
   i.size:=a.size;
   i.text:=a.text;
   i.next=0;
   getid(PCB set,receiver,j); //获得接收进程内部标识符j
   wait(j.mutex);    //申请消息队列互斥操作权
   insert(j.mg,i);   //将消息缓冲区i插入j消息队列尾部
   signal(j.mutex);  //释放消息队列互斥操作权
   signal(j.sm);     //释放消息队列资源信号量
end
```

将a发送区的消息拷贝至消息缓冲区i中

§6 进程高级通信

- 发送原语形式化描述

```
procedure receive(b)
 begin
 j:=internal name;  //获得自己的内部标识符j
 wait(j.sm);  //申请一个可接收的消息
 wait(j.mutex);  //申请消息队列互斥操作权
 remove(j.mq,i);  //从消息队列中移出队首消息i
 signal(j.mutex); //释放消息队列互斥操作权
 b.sender:=i.sender;
 b.size:=i.size;
 b.text:=i.text;
 releasebuf(i)
 end
```

将消息缓冲区i中的消息拷贝到接收区b中

§6 进程高级通信

例1：消息缓冲队列通信中的临界资源是 C 。

A. 信箱　　B.队列中的某个消息缓冲区

C.整个消息缓冲队列　　D.无临界资源存在

例2：为了实现消息缓冲通信，进程控制块中必须设置和实现与这一通信有关的数据项，它们分别是：消息队列队首指针，和消息队列互斥信号量，和消息队列中消息数信号量。