

进程同步机构

什么是锁?

用变量 w 代表某种资源的状态, w 称为“锁”。

上锁操作和开锁操作

- 检测 w 的值 (是0还是1);
 - 如果 w 的值为1, 继续检测;
 - 如果 w 的值为0, 将锁位置1 (表示占用资源), 进入临界区执行。
(此为上锁操作)
-
- 临界资源使用完毕, 将锁位置0。
(此为开锁操作)

① 上锁算法Lock

输入：锁变量w

输出：无

```
{  
    while (w==1); /* 测试锁位的值*/  
    w=1;          /*上锁*/  
}
```

Q：这个算法有什么特点？

② 开锁算法Unlock

输入：锁变量w

输出：无

```
{  
    w=0; /*开锁*/  
}
```

利用等待和唤醒原语实现上锁和解锁

(1) 上锁算法Lock

```
while (w!=0) {  
    将当前进程送入等待队列;  
    进程状态变为等待态;  
    转进程调度;  
}  
w=1;
```

(2) 开锁算法Unlock

```
if (w等待队列不为空)  
    唤醒等待队列中的进程;  
w=0;
```

用锁实现进程互斥

```
main()
{
    lock w=0;    /* 互斥锁 */
    cobegin
        pa();
        pb();
    coend
}
```

```
pa()
{
    :
    lock(w) ;
    CSa ;
    unlock(w)
    :
}
```

```
pb()
{
    :
    lock(w) ;
    CSb ;
    unlock(w)
    :
}
```

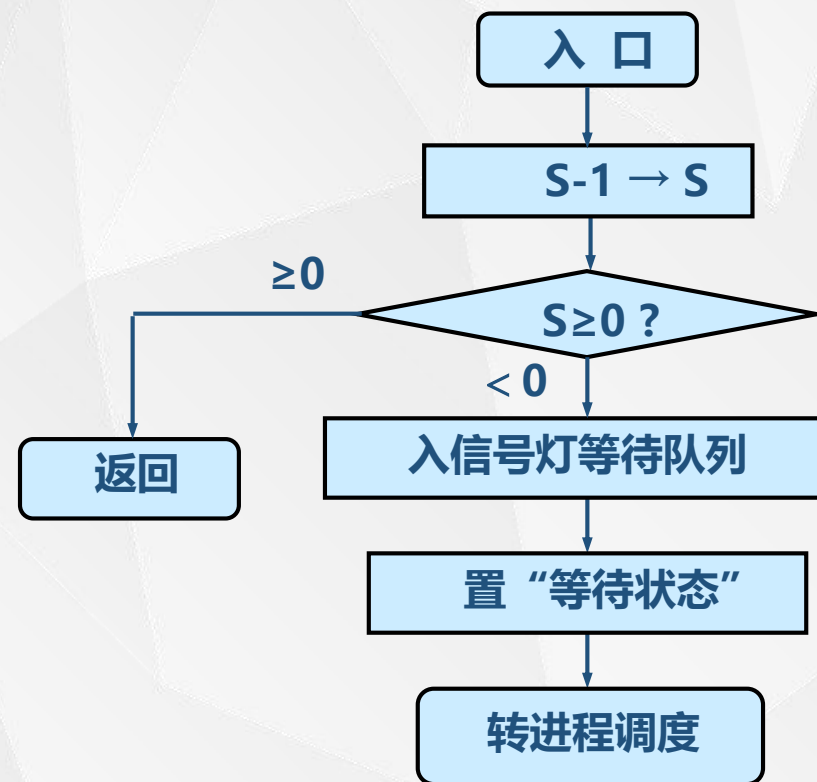
- 1965年由荷兰学者Dijkstra提出的进程同步机制，其基本思想是用一种新的变量类型——信号灯（semaphore）来表示当前资源的可用数量。
- 信号灯是一个确定的二元组 (s, q) ， s 是一个具有非负初值的整型变量， q 是一个初始状态为空的队列。操作系统利用信号灯对并发进程和共享资源进行控制和管理。
 - 变量值 $s \geq 0$ 时，表示绿灯，进程执行；
 - 变量值 $s < 0$ 时，表示红灯，进程停止执行。

注意：创建信号灯时应说明信号灯的意义和 s 的初值，且初值绝不能为负值。

- P和V是对信号灯变量可以执行的操作，分别代表荷兰语的test(proberen)和increment(verhogen)。

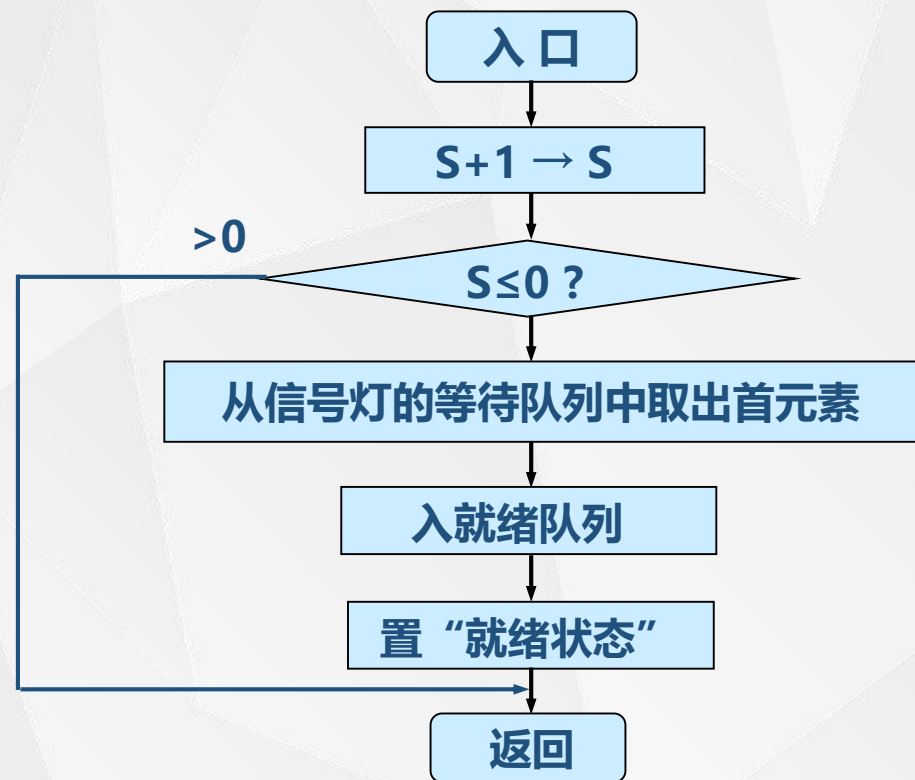
对信号灯s的P操作记为**P(s)**。

P(s)是一个不可分割的原语操作，即取信号灯值减1，若相减结果为负，则调用P(s)的进程被阻，并插入到该信号灯的等待队列中，否则该进程继续执行。



对信号灯s的V操作记为**V(s)**。

V(s)是一个不可分割的原语操作，即取信号灯值加1，若相加结果大于零，进程继续执行，否则，唤醒在该信号灯等待队列上的第一个进程。



用信号灯实现进程互斥

```
main( )
{
    semaphore mutex=1;    /* 互斥信号灯 */
    cobegin
        pa( );
        pb( );
    coend
}

pa( )                pb( )
{
    :
    p(mutex);
    CSa ;
    v(mutex);
    :
}

{
    :
    p(mutex);
    CSb ;
    v(mutex);
    :
}
```

信号灯可能的取值

两个并发进程，互斥信号灯的值仅取1、0和-1三个值。

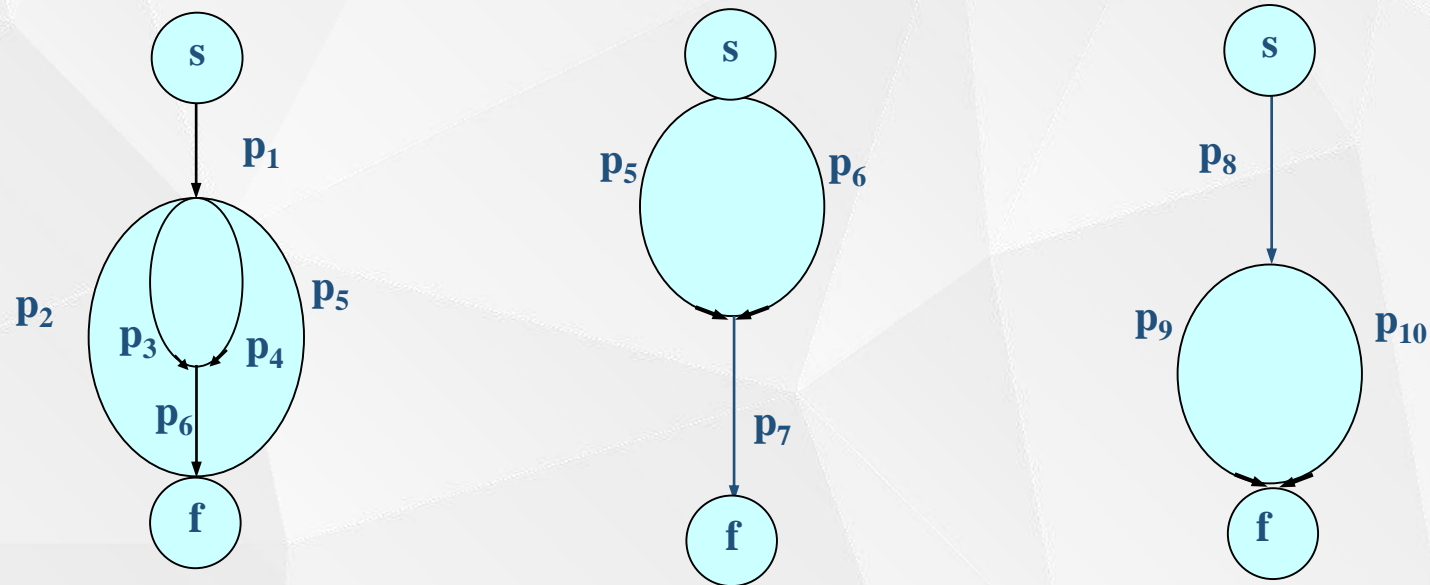
- 1：表示没有进程进入临界区；
- 0：表示有一个进程进入临界区；
- -1：表示一个进程进入临界区，另一个进程等待进入。

讨论：若有 n 个并发进程需要互斥

- 互斥信号灯 s 的取值范围是什么？
- 若 $s=1$ ，代表什么含义？
- 若 $s=0$ ，代表什么含义？
- 若 $s=-k$ ，代表什么含义？

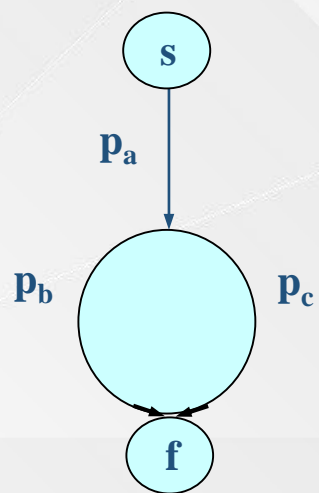
- **信号灯的初值不能为负值。**
- **必须成对使用P和V操作：**遗漏P操作则不能保证互斥访问，遗漏V操作则不能在使用临界资源之后将其释放（给其他等待的进程）。
- **P、V操作不能次序错误、重复或遗漏。**

利用信号灯实现同步——合作进程的执行次序



进程流图

例：pa、pb、pc为一组合作进程，其进程流图如图所示，试用信号灯实现这三个进程的同步。



分析任务的同步关系

任务启动后 p_a 先执行，当它结束后， p_b 、 p_c 可以开始执行， p_b 、 p_c 都执行完毕后，任务终止。

信号灯设置

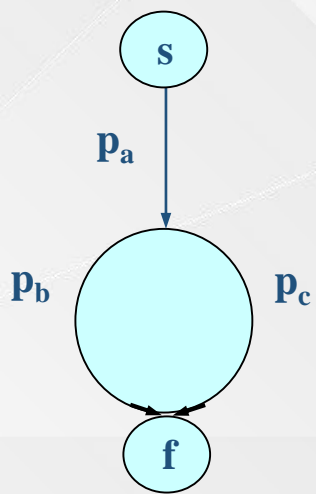
设两个同步信号灯 s_b 、 s_c 分别表示进程 p_b 和 p_c 能否开始执行，其初值均为0。

同步描述

p_a :
:
 $v(s_b);$
 $v(s_c);$

p_b :
 $p(s_b);$
:
:

p_c :
 $p(s_c);$
:
:



讨论：能否用一个信号灯实现？

```

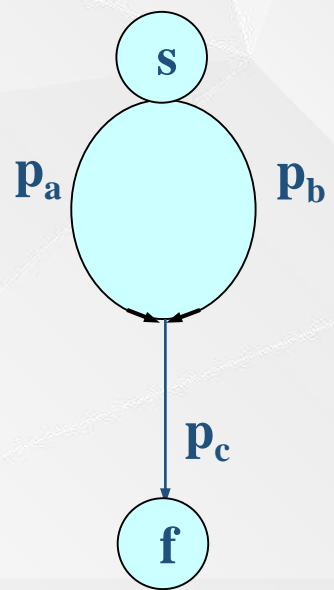
main( ) {
    semaphore sb=0; /*pb进程能否开始执行*/
    semaphore sc=0; /*pc进程能否开始执行*/
    cobegin
        pa( );
        pb( );
        pc( );
    coend
}

pa( ) {
    ⋮
    v(sb);
    v(sc);
}

pb( ) {
    p(sb);
    ⋮
    ⋮
}

pc( ) {
    p(sc);
    ⋮
    ⋮
}

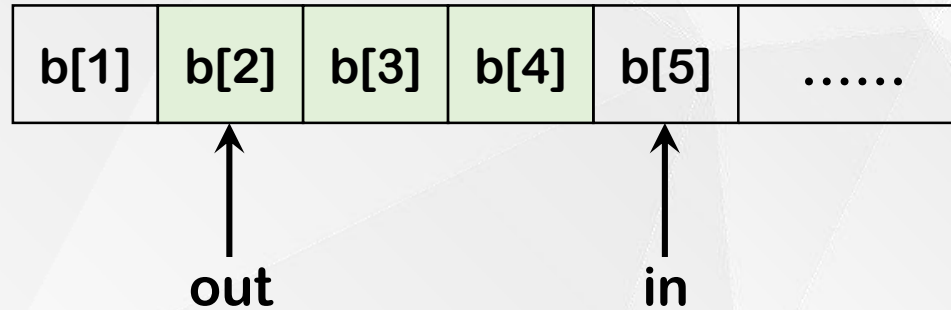
```

一到多个生产者和一到多个消费者共享一个可同时容纳多个产品的缓冲区，生产者不断生产产品并放入缓冲区，消费者不断从缓冲区取出产品并消耗它。 要求：

- 任何时刻最多只能有一个生产者或消费者访问缓冲区；
- 禁止生产者向满缓冲区输送产品；
- 禁止消费者从空缓冲区提取产品。

缓冲区无界的情况



producer:

```
while(true) {  
    /* produce item v */  
    b[in] = v;  
    in++;  
}
```

consumer:

```
while(true) {  
    while (in <= out);  
    w = b[out];  
    out++;  
    /* consume item w */  
}
```

Q: 缓存区无界时需要考虑哪些并发问题?

1. 互斥;
2. 有没有产品?
3. 没有产品可用时, 消费者应该被阻塞。

方案一

```
int n=0;           //可用产品的数量
semaphore s = 1;   //互斥信号量
semaphore delay = 0; //同步信号量
```

```
void producer()
{
    while (true) {
        生产产品;
        P(s);
        放入一个产品;
        n++;
        if (n==1) V(delay);
        V(s);
    }
}
```

```
void consumer()
{
    P(delay);
    while (true) {
        P(s);
        取出一个产品;
        n--;
        V(s);
        消费产品;
        if (n==0) P(delay);
    }
}
```

Q: 有什么问题?

	Producer	Consumer	s	n	delay
1			1	0	0
2	P(s)		0	0	0
3	n++		0	1	0
4	if (n==1) V(delay);		0	1	1
5	V(s)		1	1	1
6		P(delay)	1	1	0
7		P(s)	0	1	0
8		n--	0	0	0
9		V(s)	1	0	0
10	P(s)		0	0	0
11	n++		0	1	0
12	if (n==1) V(delay);		0	1	1
13	V(s)		1	1	1
14		if (n==0) P(delay);	1	1	1
15		P(s)	0	1	1
16		n--	0	0	1
17		V(s)	1	0	1
18		if (n==0) P(delay);	1	0	0
18		P(s)	0	0	0
20		n--	1	-1	0

方案二

```
int n=0;  
semaphore s = 1, delay = 0;
```

```
void producer()  
{  
    while (true) {  
        生产产品;  
        P(s);  
        放入一个产品;  
        n++;  
        if (n==1) V(delay);  
        V(s);  
    }  
}
```

```
void consumer()  
{  
    P(delay);  
    while (true) {  
        P(s);  
        取出一个产品;  
        n--;  
        if (n==0) P(delay);  
        V(s);  
        消费产品;  
    }  
}
```

Q: 有什么问题?

方案三

```
int n=0;  
semaphore s = 1, delay = 0;
```

```
void producer()  
{  
    while (true) {  
        生产产品;  
        P(s);  
        放入一个产品;  
        n++;  
        if (n==1) V(delay);  
        V(s);  
    }  
}
```

```
void consumer()  
{  
    int m;  
    P(delay);  
    while (true) {  
        P(s);  
        取出一个产品;  
        n--;  
        m=n;  
        V(s);  
        消费产品;  
        if (m==0) P(delay);  
    }  
}
```

方案四

```
semaphore s = 1; //互斥
```

```
semaphore n = 0; //缓冲区中的产品数量
```

```
void producer()
```

```
{
```

```
    while (true) {
```

```
        生产产品;
```

```
        P(s);
```

```
        放入一个产品;
```

```
        V(n);
```

```
        V(s);
```

```
    }
```

```
}
```

```
void consumer()
```

```
{
```

```
    while (true) {
```

```
        P(n);
```

```
        P(s);
```

```
        取出一个产品;
```

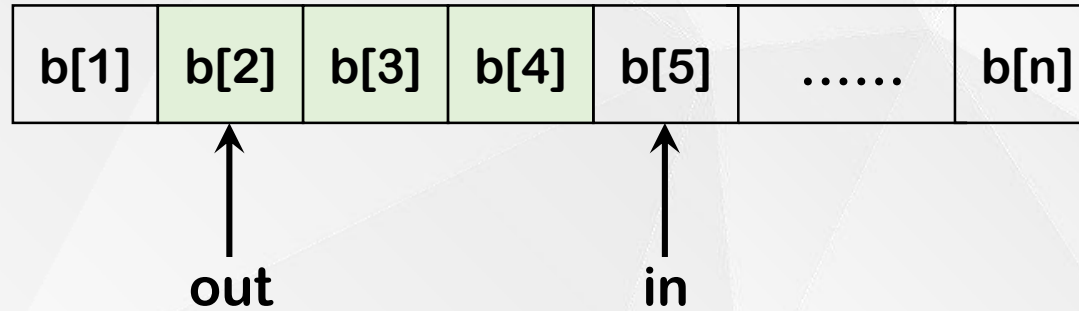
```
        V(s);
```

```
        消费产品;
```

```
    }
```

```
}
```

缓冲区有界的情况



producer:

```
while(true) {  
    /* produce item v */  
    while ( (in+1)%n == out );  
    b[in] = v;  
    in = (in+1)%n;  
}
```

consumer:

```
while(true) {  
    while (in == out);  
    w = b[out];  
    out = (out+1)%n;  
    /* consume item w */  
}
```


缓存区有界时需要考虑的并发问题：

1. 互斥
2. 有没有空缓冲区？没有可用的空缓冲区时，生产者应该被阻塞
3. 有没有产品（满缓冲区）？没有可消费的产品时，消费者应该被阻塞


如何设置信号量？

1. 一个互斥信号量
mutex：表示有界缓冲区是否被占用，初值 = 1
2. 两个同步信号量
 s_a ：表示满缓冲区的数目，初值 = 0
 s_b ：表示空缓冲区的数目，初值 = n


生产者:




```
生产;  
p(sb);  
p(mutex);  
将数据放入有界缓冲区;  
v(mutex);  
v(sa);
```



消费者:



```
p(sa);  
p(mutex);  
从有界缓冲区中取数据;  
v(mutex);  
v(sb);  
消费;
```



讨论：下面这个流程有没有问题？

生产者:

生产;

$p(mutex);$

$p(s_b);$

将数据放入有界缓冲区;

$v(mutex);$

$v(s_a);$

消费者:

$p(mutex);$

$p(s_a);$

从有界缓冲区中取数据;

$v(mutex);$

$v(s_b);$

消费;


```
main( )
{
    semaphore mutex=1; /*对有界缓冲区进行操作的互斥信号灯*/
    semaphore sa=0;      /*满缓冲区的数目*/
    semaphore sb=n;      /*空缓冲区的数目*/
    cobegin
        p1( ); p2( ); ... pm( );
        c1( ); c2( ); ... ck( );
    coend
}
```

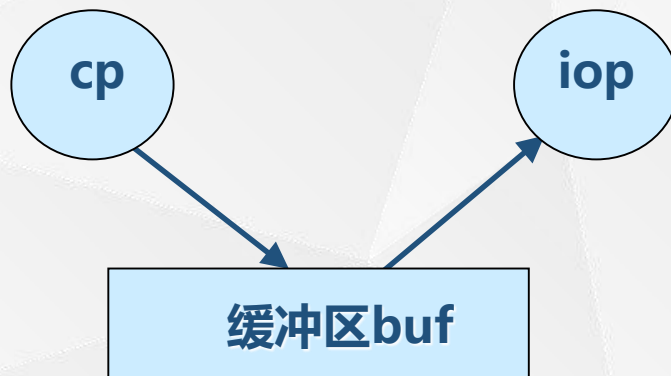
```
pi()  
{  
    while(生产未完成)  
    {  
        :  
        生产一个产品;  
        p(sb);  
        p(mutex);  
        送一个产品到有界缓冲区;  
        v(mutex);  
        v(sa);  
    }  
}
```

```
cj()  
{  
    while(还要继续消费)  
    {  
        p(sa);  
        p(mutex);  
        从有界缓冲区中取产品;  
        v(mutex);  
        v(sb);  
        消费一个产品;  
        :  
    }  
}
```

讨论：这些并发进程的瓶颈在哪里？

生产者-消费者问题举例——誊抄

计算进程 cp 和打印进程 iop 共用一个缓冲区，为了完成正确的计算与打印，试用信号量的 p、v 操作实现这两个进程的同步。



两个进程的任务

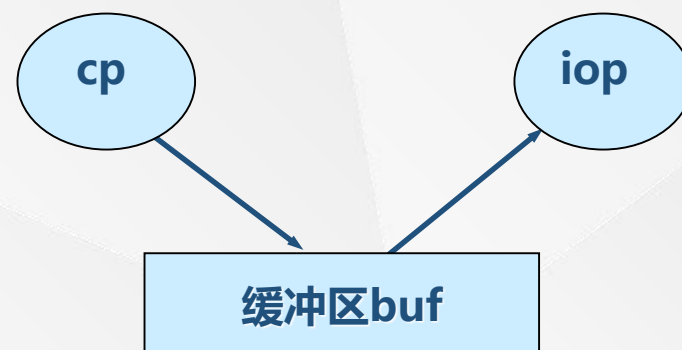
- 计算进程 cp 经过计算，将计算结果送入 buf。
- 打印进程 iop 把 buf 中的数据取出，然后打印。

任务的同步关系

- cp进程把计算结果送入buf后，iop进程才能从buf中取出结果去打印，否则必须等待。
- iop进程把buf中的数据取出后，cp进程才能把下一个计算结果数据送入buf中，否则必须等待。

信号量如何设置？

- S_{full} : 表示缓冲区是否有可打印的数据，其初值为0。
- S_{empty} : 表示缓冲区是否空闲，其初值为1。
- 需要设置互斥信号量吗？



信号灯 s_{full} 和 s_{empty} 如何使用?

cp:

计算一个数据;

$p(s_{empty});$

将数据放入buf ;

$v(s_{full});$

iop:

$p(s_{full});$

从buf中取 数据;

$v(s_{empty});$

打印;

讨论: 两个信号量可能的取值及其所表示的状态。

```

main( )
{
    semaphore sfull =0;    /*表示缓冲区中是否有数据*/
    semaphore sempty =1;  /*表示缓冲区是否为空*/
    cobegin
        cp( ); iop( );
    coend
}

cp( )
{
    while(计算未完成)
    {
        得到一个计算结果;
        p(sempty);
        将数送到缓冲区中;
        v(sfull);
    }
}

iop( )
{
    while(打印工作未完成)
    {
        p(sfull);
        从缓冲区中取一个数;
        v(sempty);
        从打印机上输出;
    }
}

```


思考题：

某商场有一个地下车库，有 N 个停车位，车辆只能通过一个指定的通道进出该车库，通道处只能容一辆车通过。请设计信号灯和P、V操作，给出车辆进库及出库的程序描述。

分析同步关系：

- 所有车辆必须互斥使用通道
- 车库里有空的停车位时车辆才能进库，否则等待
- 车辆出库后应该通知正在等待停车位的车辆

```
main()
{
    semaphore mutex=1; //互斥信号量
    semaphore n=N;     //空位数量
    cobegin
        p1(); p2(); .....pm();
    coend
}
```

```
pi()
{
    .....
    p(n);
    p(mutex);
    进入车库;
    v(mutex);
    停车;
    p(mutex);
    开出车库;
    v(mutex);
    v(n);
    .....
}
```

思考题：

某公园有一个长凳，其上最多可以坐5个人。游客按以下规则使用长凳：

- (1) 如果长凳上还有空间可以坐，就坐到长凳上休息，直到休息结束，离开长凳。
- (2) 如果长凳上没有空间，就转身离开。

```
main()
{
    semaphore mutex=1; //互斥信号量
    semaphore n=5;      //长凳上的空位数
    cobegin
        p1(); p2(); .....pn();
    coend
}
```

对吗?

```
pi()
{
    p(n);
    p(mutex);
    在长凳上找位置坐下;
    v(mutex);
    休息;
    p(mutex);
    离开长凳;
    v(mutex);
    v(n);
}
```

```
main()
{
    int n=5;
    semaphore mutex=1; //互斥信号量
    cobegin
        p1(); p2(); .....pm();
    coend
}
```

```
pi()
{
    p(mutex);
    if (n==0) {
        v(mutex);
        转身离开;
    }
    else {
        n--;
        在长凳上找位置坐下;
        v(mutex);
        休息;
        p(mutex);
        n++;
        离开长凳;
        v(mutex);
    }
}
```