Q1：另类PV操作问题

有一个系统，定义P、V操作如下：

P(s):

s.count --;

if s<0 then

将本进程插入相应队列末尾等待；

V(s)：

s.count ++;

if s<=0 then

从相应等待队列末尾唤醒一个进程，将其插入就绪队列；

思考并回答：

a. 这样定义PV操作是否有问题？**有问题，后进先出**

b. 用这样的PV操作实现N个进程竞争使用某一共享变量的互斥机制。

c. 对于b的解法，有无效率更高的方法。如有，试问降低了多少复杂性？

漏斗法

思路：

- 令每个信号量上的等待队列中始终只有一个进程

- N个进程至多有N-1个等待
- 设置N-1个信号量：S[0] ~ S[N-2]
- 信号量的值从1到N-1

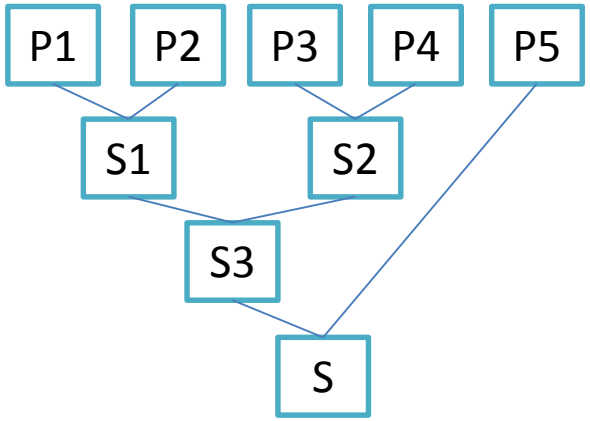
S[0]	S[1]	S[2]	.....	S[N-2]
1	2	3	.....	N-1

```
Procedure_i() {
        int j;
        for(j = N - 2; j >= 0; j --)
            P(S[j]);
        使用资源;
        for(j = 0; j <= N - 2; j++)
            V(S[j]);
}
```

二叉树法

思路：

- 令每个信号量上的等待队列中始终只有一个进程
- 相邻的两个进程两两竞争，优胜者进入下一轮竞争，最后得出优胜者使用资源（ $\lceil logN \rceil$ ）




```
Procedure_1/2() {
    P(S1);
    P(S3);
    P(S);
    使用资源;
    V(S);
    V(S3);
    V(S1);
}
```

```
Procedure_3/4() {
    P(S2);
    P(S3);
    P(S);
    使用资源;
    V(S);
    V(S3);
    V(S2);
}
```

```
Procedure_5() {
    P(S);
    使用资源;
    V(S);
}
```

Q2：睡眠理发师问题

理发店里有一位理发师，一把理发椅和N把供等候理发的顾客坐的椅子。

如果没有顾客，则理发师便在理发椅上睡觉。

当一个顾客到来时，他必须先唤醒理发师；如果顾客到来时理发师正在理发，则如果有空椅子，可坐下来等；否则离开。

试用P、V操作解决睡眠理发师问题。

步骤1 设定信号量和变量

1. 什么时候等？

客人：有椅子的时候，理发师在剪发　　semaphore barber

理发师：没有客人在座位上　　semaphore customer

2. 不等，但要判断？

客人：没有椅子剩下的时候，走掉　　int waiting

步骤2 写出框架

Barber:

```
while(true) {
        P(customer); // 若没有customer，则睡觉
        waiting --;
        V(barber);
        理发;
}
```

Customer:

```
if (waiting < N) {
        waiting ++;
        V(customer);
        P(barber);
        被理发;
} else {
}
```

```
semaphore customer = 0;
semaphore  barber = 0;
int waiting = 0;
```

// 等待的customer
// 理发请求，为0则睡觉

步骤3 加互斥

Barber:

```
while(true) {
        P(customer); // 若没有customer，则睡觉
        P(M);
        waiting --;
        V(M);
        V(barber);
        理发;
}
```

Customer:

```
P(M);
if (waiting < N) {
        waiting ++;
        V(M);
        V(customer);
        P(barber);
        被理发;
} else {
        V(M);
}
```

```
semaphore customer = barber = 0;
semaphore M = 1;
int waiting = 0;
```

理发师问题变化

理发店里有一位理发师，一把理发椅和N把供等候理发的顾客坐的椅子。

如果没有顾客，则理发师便在理发椅上睡觉。

当一个顾客到来时，他必须先唤醒理发师；如果顾客到来时理发师正在理发，则如果有空椅子，可坐下来等；否则**等待椅子**。

试用P、V操作解决睡眠理发师问题。

步骤1 设定信号量和变量

1. 什么时候等？

客人：有椅子的时候，理发师在剪发　　semaphore barber

理发师：没有客人在座位上　　semaphore customer

客人：没有椅子的时候，等待椅子　　semaphore seats

步骤2 写出框架

Barber:

```
while(true) {
        P(customer); // 若没有customer，则睡觉
        V(barber);
        理发;
}
```

Customer:

```
P(seats);
V(customer);
P(barber);
V(seats);
被理发;
```

```
semaphore customer = 0;     // 等待的customer
semaphore barber = 0;       // 理发请求，为0则睡觉
semaphore seats = N;        // 剩余椅子数
```

Q3：复杂的消息缓冲问题

消息缓冲区为k个，有m个发送进程，n个接收进程，每个接收进程对发送来的消息都必须取一次。

试用P、V操作解决发送进程和接受进程之间的正确通信问题。

步骤1 设定信号量和变量

1. 什么时候等？

发送者：缓冲区满　　semaphore send[k]

接收者：缓冲区空　　semaphore receive[k]

2. 计数变量

k个缓冲区：缓冲区指针　　int cur

每个缓冲区需要接收n次：计数　　int count[k]

步骤2 写出框架

Sender:

```
while(true) {
        cur = (cur + 1) % k;
        P(send[cur]);
         写入内容;
        V(receive[cur]);
}
```

```
semaphore send[k] = {1};
semaphore receive[k] = {0};
int cur = 0, count[k] = {0};
```

Receiver:

```
for (int i = 0; i < k; i ++) {
        count[i]++;
        if (count[i] == 1)
           P(receive[i]);

        读取内容;

        if (count[i] == n) {
           count[i] = 0;
           V(send[i]);
        }
}
```

步骤3 加互斥

Sender:

```
while(true) {
    P(mutex1);
        cur = (cur + 1) % k;
        P(send[cur]);
         写入内容;
        V(receive[cur]);
    V(mutex1);
}
```

Receiver:

```
for (int i = 0; i < k; i ++) {
        P(mutex[i]);
            count[i]++;
            if (count[i] == 1)
                P(receive[i]);
        V(mutex[i]);
            读取内容;
        P(mutex[i]);
            if (count[i] == n2) {
                count[i] = 0;
                V(send[i]);
            }
        V(mutex[i]);
}
```

```
semaphore send[k] = {1}, receive[k] = {0};
semaphore mutex1 = 1, mutex[k] = {1};
int cur = 0, count[k] = {0};
```

Q4：读者写者问题

要求满足条件：

允许多个读者同时执行读操作

不允许多个写者同时操作

不允许读者、写者同时操作

第一类读者写者问题（读者优先）

第二类读者写者问题（写者优先）

第三类读者写者问题？

维基百科：Readers–writers problem

http://en.wikipedia.org/w/index.php?title=Readers%E2%80%93writers_problem

第一类读者写者问题（读者优先）

Reader :

```
while (TRUE) {
        P(mutex1);
          rc = rc + 1;
          if (rc == 1)  P (w);
        V(mutex1);

        读操作

        P(mutex1);
          rc = rc - 1;
          if (rc == 0)  V(w);
        V(mutex1);
}
```

Writer :

```
while (TRUE) {
        P(w);

        写操作

        V(w);
}
```

有什么问题？

写者饥饿

```
semaphore mutex1 = 1;
semaphore w = 1;
int rc = 0;
```

第三类读者写者问题

Reader :

```
while (TRUE) {
        P(r);
          P(mutex1);
           rc = rc + 1;
           if (rc == 1)  P (w);
          V(mutex1);
        V(r);

        读操作

        P(mutex1);
          rc = rc - 1;
          if (rc == 0)  V(w);
        V(mutex1);
}
```

Writer :

```
while (TRUE) {
        P(r);
        P(w);
        V(r);

        写操作

        V(w);
}
```

```
P(r);
P(w);
写操作
V(w);
V(r);  ?
```

如何提高写者的优先级，实现写者优先？

```
semaphore mutex1 = 1;
semaphore w = r = 1;
int rc = 0;
```

第二类读者写者问题（写者优先）

Reader :

```
while (TRUE) {
        P(r);
          P(mutex1);
           rc = rc + 1;
           if (rc == 1)  P (w);
          V(mutex1);
        V(r);

        读操作

        P(mutex1);
          rc = rc - 1;
          if (rc == 0)  V (w);
        V(mutex1);
}
```

Writer :

```
while (TRUE) {
        P(mutex2);
          wc = wc + 1;
          if (wc == 1)  P (r);
        V(mutex2);
        P(w);

        写操作

        V(w);
        P(mutex2);
          wc = wc - 1;
          if (wc == 0)  V (r);
        V(mutex2);
}
```

```
semaphore mutex1 = mutex2 = 1;
semaphore w = r = 1;
int rc = wc = 0;
```

有没有问题？？？

第二类读者写者问题（写者优先）

Reader :

```
while (TRUE) {
        P(mutex3);
          P(r);
            P(mutex1);
              rc = rc + 1;
              if (rc == 1)  P (w);
            V(mutex1);
          V(r);
        V(mutex3);

        读操作

        P(mutex1);
          rc = rc - 1;
          if (rc == 0)  V (w);
        V(mutex1);
}
```

Writer :

```
while (TRUE) {
        P(mutex2);
          wc = wc + 1;
          if (wc == 1)  P (r);
        V(mutex2);
        P(w);

        写操作

        V(w);
        P(mutex2);
          wc = wc - 1;
          if (wc == 0)  V (r);
        V(mutex2);
}
```

```
semaphore mutex1 = mutex2 = mutex3 = 1;
semaphore w = r = 1;
int rc = wc = 0;
```

Q5：思考题1 食品销售问题

某商店有两种食品A和B， 最大数量各为m个。该商店将A、B两种食品搭配出售，每次各取一个。

为避免食品变质，遵循先到食品先出售的原则。有两个食品公司分别不断地供应A、B两种食品(每次一个)。

为保证正常销售，当某种食品的数量比另一种的数量超过k(k<m)个时，暂停对数量大的食品进货。

试用P、V操作解决上述问题中的同步和互斥关系。

步骤1 定义信号量和变量

1. 等待

公司A：

1) 商店A存货大于m时　　semaphore A
2) 商店A存货比B存货多k个　　semaphore stopA

公司B：

1) 商店B存货大于m时　　semaphore B
2) 商店B存货比A存货多k个　　semaphore stopB

2. 不等待

商店：

1) A物品无存货时　　int nA
2) B物品无存货时　　int nB

步骤2 写框架

- 商店：

```
while (true) {
        if (nA > 0 && nB > 0)
        {
                nA--;
                nB--;
                V(A);
                V(B);
        }
}
```

- 公司A

```
while (true) {
    P(A);
    V(stopB);
    P(stopA);
    nA++;
}
```

- 公司B

```
while(true) {
    P(B);
    V(stopA);
    P(stopB);
    nB++;
}
```

```
semaphore A = B = m;
semaphore stopA = stopB = k;
int nA = nB = 0;
```

步骤3 加互斥量

- 商店：

```
while (true) {
        P(Ma); P(Mb);
        if (nA > 0 && nB > 0)
        {
                nA--;
                nB--;
                V(Ma); V(Mb);
                V(A);
                V(B);
        } else {
                V(Ma); V(Mb);
        }
}
```

- 公司A

```
while (true) {
   P(A);
   V(stopB);
   P(stopA);
   P(Ma);
    nA++;
   V(Ma);
}
```

- 公司B

```
while (true) {
   P(B);
   V(stopA);
   P(stopB);
   P(Mb);
    nB++;
   V(Mb);
}
```

```
semaphore A = B = m;
semaphore Ma = Mb = 1;
semaphore stopA = stopB = k;
int nA = nB = 0;
```

Q6：思考题2 船闸问题

由于水面高度不同，有160~175米的落差，所以三峡大坝有五级船闸，T1~T5。由上游驶来的船需经由各级船闸到下游；由下游驶来的船需经由各级船闸到上游。

假设只能允许单方向通行（此假设与实际情况不符，实际为双向各五级船闸）。

试用P、V操作正确解决三峡大坝船闸调度问题。

类似读者写者问题

```
void shipA() {
        P(mutexA);
         A2B_count++;
         if (A2B_count==1)  P(lock);
        V(mutexA);

        通过船闸;

        P(mutexA);
         A2B_count--;
         if (A2B_count==0)  V(lock);
        V(mutexA);
}
```

```
void shipB() {
        P(mutexB);
         B2A_count++;
         if (B2A_count==1)  P(lock);
        V(mutexB);

        通过船闸;

        P(mutexB);
         B2A_count--;
         if (B2A_count==0)  V(lock);
        V(mutexB);
}
```

```
semaphore mutexA = mutexB = 1;
semaphore lock = 1;
int A2B_count = B2A_count = 0;
```

有什么问题？？？

会出现饥饿　如何避免？

PV操作题 - 注意事项

1. PV成对出现
2. 等待的情景要考虑周全
3. 注意互斥的范围
4. 不要尝试读信号量的值，需要计数时使用int变量
5. 信号量和变量的初始化
6. 死锁和饥饿
7. 注释！
8. 卷面清晰！