1. 若有 N 个进程共享一个临界资源，信号量取值范围如何？

答案：

（1） 没有进程访问临界区，信号量取值为 n

（2） 只有 1 个进程访问临界区，信号量取值为 n-1

（3） 有 1 个进程正在访问，i 个进程等待，信号量取值为 n-1-i

2. 利用 PV 操作实现读者-写者问题中的写者优先

答案：

要求： (1)多个读者可以同时进行读； (2)写者必须互斥(只允许一个写者写，也不能读者写者同时进行)； (3)写者优先于读者（一旦有写者，则后续读者必须等待，唤醒时优先考虑写者）。

变量 int readcount=0， writecount = 0。

设置 5 个信号量，依次为 mutexReadCount, mutexWriteCount, mutexPriority, r, w.

代码如下：

```
Reader()
   {
   /*确保写操作在 V(r)的优先性*/
   P(mutexPriority) ;
   P(r)
      P(mutexReadCount);
      readcount ++;
      //若为第一个读者，互斥写者
      if (readcount==1) P(w);
      V(mutexReadCount);
   V(r)
   V(mutexPriority);
   reading is performed….
   P(mutexReadCount);
   readcount --;
   //若当前读者为最后一个，则唤醒写者
   if (readcount==0) V(w);
   V(mutexReadCount);
}
```

```
Writer()
{
   P(mutexWriteCount);
   //写者入队列
   writecount ++;
   //若为第一个写者，阻止后续的写者
   if (writecount==1) P(r);
   V(mutexWriteCount);
   //互斥其他的写者
   P(w);
   writing is performed…
   V(w);
   P(mutexWriteCount);
   writecount --;
   if(writecount == 0) V(r);
   V(mutexWriteCount);
}
```

Reference: Communications of the ACM: Concurrent Control with "Readers" and "Writers" P.J. Courtois,* F. H, 1971

扩展问题：如何使得读写都公平：

semaphores: no_writers, no_readers, counter_mutex (initial value is 1)

shared variables: nreaders (initial value is 0)

local variables: prev, current

```
Read（）
   {
   P( no_writers );
   P( counter_mutex );
   prev = nreaders;
   nreaders++;
   V( counter_mutex );
   if (prev== 0)   P( no_readers );
   V( no_writers );
   ... read ...
   P( counter_mutex );
   nreaders--;
   current = nreaders;
   V( counter_mutex );
   if (current = =0) V( no_readers );
}
```

```
Write()
{
      P( no_writers );
      P( no_readers );
      V( no_writers );
      ...   write ...
      V( no_readers );
}
```

3-7 答案：

本题中使用一个信号量 m 用于互斥过河。

同步算法描述如下：

P(m)；

过河；

V(m)；

若扩展本题目，允许同向的多辆车通行，即要么左道的多辆车都走，要么右道的多辆车都走，则问题转化为类似于读者写者问题中读者优先问题的变型

本题中使用三个信号量：

mutexl、mutexr 用于互斥访问共享变量 countl 及 countr，初值均为 1。

wait 用于申请过桥，初值也为 1。

同步算法描述如下

semaphore mutexl=1；

semaphore mutexr=1；

semaphore wait=1；

int countl=0；

Int countr=0；

```
main()
{
  cobegin
    passl()；
    passr()；
    coend
}
```

```
passl()
{
      P(wait)；
      P(mutexl)；
      countl++；
      if(countl==1)P(mutexr)；
      V(mutexl)；
      V(wait)；
      过河；
      P(mutexl)；
      countl- -；
      if(countl==0) V(mutexr)；
        V(mutexl)；
}
```

```
passr()
{
      P(wait)；
      P(mutexr)；
      countr++；
      if(countr==1) P(mutexl)；
      V(mutexr)；
      V(wait)；
      过河；
      P(mutexr)；
      countr- -；
      if (countr==0) V(mutexl)；
      V(mutexr)；
}
```

3(10): 这是一个典型的利用信号量机制，建立进程同步关系的例子

本题中使用 4 个信号量：

(1)S1 表示是否可以开始点菜，初值为 1

(2)S2 表示是否可以开始做菜，初值为 0

(3)S3 表示是否可以开始打包，初值为 0

(4)S4 表示是否可以提交食品，初值为 0

同步算法描述如下：

```
semaphore S1=1；
semaphore S2=0；
semaphore S3=0；
semaphore S4=0；
main()
{
   cobegin
        LB（）；
        CS（）；
        DBG（）；
        CNY（）；
     Coend
```

```
}
LB()
{
    while(true)
     {
         顾客到达；
         p(S1)；
         接受顾客点菜；
         v(S2)；
      }
  }

CS()
{    while(true)
     {
         p(S2)；
         准备顾客的饭菜；
         v(S3)；
      }
}
DBG()
{    while(true)
     {
         p(S3)；
         打包顾客的饭菜；
         v(S4)；
      }
}
CNY()
{    while(true)
     {
         p(S4)；
         收款并提交食品；
         v(S1)；
      }
}
```

补充 1：Jurassic 公园有一个恐龙博物馆和一个花园，有 m 个旅客和 n 辆车，每辆车仅能乘一个旅客。旅客在博物馆逛了一会，然后，排队乘坐旅行车，当一辆车可用时，它载入一个旅客，再绕花园行驶任意长的时间。若 n 辆车都已被旅客乘坐游玩，则想坐车的旅客需要等待。如果一辆车已经空闲，但没有游玩的旅客了，那么，车辆要等待。试用信号量和 P、V 操作同步 m 个旅客和 n 辆车子。

答：这是一个同步问题，类似理发师问题，有两类进程：顾客进程和车辆进程，需要进行同步，即顾客要坐进车后才能游玩，如果车辆不足，则等待，若乘客不足则车等待，开始时让车辆进程进入等待状态。

设置 4 个信号量：

EmptyBus: 可用空车数量，初值为 n

ReadyCustomer：准备上车游玩的顾客数量，初值为 0

ReadyCar：为乘客已经准备好了车辆，初值为 0

mutex：互斥共享区域，初值为 1

sharearea:一个登记车辆\被服务乘客信息的共享区;

```
Customer()
{
   P(EmptyBus);//若空车数不足等待
   P(mutex);
   在共享区 sharearea 登记被服务的顾客的信息
   V(ReadyCustomer);//唤醒一辆车
   V(mutex);
   乘客上车
   P(ReadyCar);

 }
```

```
Bus()
{
   P(ReadyCustomer);//若无顾客等待
   P(mutex);
   在共享区 sharearea 登记那一辆车被使用
   V(EmptyBus);//唤醒等待的乘客
   V(mutex);
   V(ReadyCar);
   开车为游客服务
}
```

补充 2：有 P1、P2、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。用信号量和 P、V 操作给出方案。

答：

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，我们采用读者优先策略。

rmutex,wmutex:读写互斥

Readcount: 计数互斥

```
P1()
{
   P(rmutex);
   Readcount++;
   If(Readcount ==1)P(wmutex);
   V(rmutex);
   ReadFile…
   P(rmutex);
   Readcount--;
   If(Readcount == 0)V(wmutex);
   V(rmutex);
}
```

```
P2()
{
  P(wmutex);
  WriteFile…
  V(wmutex);
}
```

```
P3()
{
   P(rmutex);
   Readcount++;
   If(Readcount ==1)P(wmutex);
   V(rmutex);
   ReadFile…
   P(rmutex);
   Readcount--;
   If(Readcount == 0)V(wmutex);
   V(rmutex);
   P(wmutex);
   WriteFile…
   V(wmutex);
 }
```

补充 3：Dijkstra 临界区软件算法描述如下，

```
enum {idle, want-in, in_cs} flag[n];
int turn;
void proc(int i){
   int j;
   while(1){
       do{
         flag[i] = want_in;
        while (turn <>i)
        {
              if    (flag[turn]==idle)
                 turn = i;
         }
         flag[i]=in_cs;
         j=0;
         while(j<n && ( j==i || flag[j]<>in_cs))
                 j++;
       }while (j<n)
       进入临界区
       flag[i]=idle;
       …
     }
```

```
    }
```

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：

```
do {
        flag[i]=want_in;                          ①
        while (turn <>i)                          ②
          if (flag[turn]==idle) turn=i;           ③
        flag[i]=in_cs;                            ④
        j=0;
        while    (j<n)&(j==i or flag[j]<>in_cs)   ⑤
                 j++;                             ⑥
}while(j<n)
        进入临界区
        flag[i]=idle;                             ⑦
```

…

(1) 满足空闲让进，忙则等待

当所有的 Pj 都不在临界区中，满足 turn ==I 并且 flag[j]≠in_cs (对于所有 j，j≠i)条件时，Pi 才能进入它的临界区，而且进程 Pi 不会改变除自己外的其他进程所对应的 flag[j]的值。另外，进程 Pi 总是先置自己的 flag[i]为 in_cs 后，才去判别 Pj 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，满足空闲让进，忙则等待。

(2) 满足有限等待

由于任何一个进程 Pi 在执行进入临界区代码时先执行语句①，其相应的 flag[i]的值不会是 idle。注意到 flag[i]==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 P0 不工作，所以，flag[turn]==flag[0]==idle。但是若干个其他进程是可能同时交替执行的，假设让进程 Pj(j=1,2,..,n-1)交错执行语句①后(这时 flag[j]=want_in)，再做语句②(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③，努力使自己的 turn 为 j。但 turn 仅有一个值，该值为最后一个执行此赋值语句的进程号，设为 k、即 turn=k(1≤k≦n-1)。接着，进程 Pj(j=1,2,..,n-1)交错执行语句④，于是最多同时可能有 n-1 个进程处于 in_cs 状态，但不要忘了仅有一个进程能成功执行语句④，将 turn 置为自己的值。

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

补充题 4: 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，并说明所定义的信号量含义。

答案：
该题为生产者、消费者的一种变形，这里有一个生产者 P1，两个消费者 P2、P3，并且要求满足如下的同步要求
（1） P1, P2, P3 必须互斥的访问缓冲区
（2） 当缓冲区空，必须 P1 来投入正整数，而 P2,P3 处于阻塞状态
（3） 当缓冲区满，必须 P2,P3 之一取出整数，而 P1 处于阻塞
设置信号量：
empty：初值为 n;
odd：初值为 0;
even：初值为 0

```
P1()
{
   While(1){
      Number = produce();
      P(empty);
      P(mutex);
      Put();
      V(mutex);
      If (number%2==0)V(even);
         Else V(odd);
   }
```

```
P2()
{
   While(1){
      P(odd);
      P(mutex);
      getodd();
      V(mutex);
      V(empty);
      Countodd();
   }

}
```

```
P3()
{
   While(1){
      P(even);
      P(mutex);
      geteven();
      V(mutex);
      V(empty);
      counteven();
   }

}
```