1. 读者写者问题(写者优先) : 1)共享读; 2)互斥写、读写互斥; 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)。

```
Semaphore RWBlock;
Semaphore Wpri;
Semaphore ReadBlock;
Semaphore Rmutex;
Semaphore Wmutex;
int readnum=0;
int writenum=0;
Read:
    while(true){
        P(Wpri);
        P(Rmutex);
        if(readnum==0){
            P(ReadBlock);
            P(RWBlock);
        }
        readnum++;
        V(Rmutex);
        V(Wpri);
        read();
        P(Rmutex);
        readnum--;
        if(readnum==0){
            V(ReadBlock);
            V(RWBlock);
        }
        V(Rmutex);
    }

Writer:
    while(true){
       P(Wmutex);
       if(writenum==0){
            P(Wpri);
            P(ReadBlock);
       }
       writenum++;
       V(Wmutex);
       P(RWBlock);
       write();
       V(RWBlock);
       P(Wmutex);
       writenum--;
       if(writenum==0){
            V(ReadBlock);
       }
       V(Wmutex);
       V(Wpri);
```

```
    }
```

2. 寿司店问题。假设一个寿司店有 5 个座位，如果你到达的时候有一个空座位，你可以立刻就坐。但是如果你到达的时候 5 个座位都是满的有人已经就坐， 这就意味着这些人都是一起来吃饭的，那么你需要等待所有的人一起离开才能就坐。 编写同步原语，实现这个场景的约束。

```
Semaphore mutex=1; //保护waiting,eating,flag.
Semaphore block=0;
int eating=0;
int waiting=0;
int flag=0;         //标记是否已经满了五个人
while(true){
    P(mutex);
    if(flag==1){
        waiting++;
        V(mutex);
        P(block);
    }else{
        eating+=1;
        if(eating==5){
            flag=1;
        }else{
            flag=0;
        }
        V(mutex);
    }
    //do some eating
    P(mutex);
    eating-=1;
    if(eating==0){
        int next=min(waiting,5);
        waiting-=n;
        eating+=n;
        if(eating==5){
            flag=1;
        }else{
            flag=0;
        }
        while(n--){
            V(block);
        }
    }
    V(mutex);

}
```

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()送入缓冲区某一个空单元中；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的含义。要求用伪代码描述。

```
Semaphore mutex=1;         //对缓冲区的访问限制
Semaphore empty=N;         //缓冲区的个数限制
Semaphore odd=0;           //已经放入的奇数个数
Semaphore even=0;          //已经放入的偶数个数

P1:
    while(true){
        int number=produce();
        P(empty);
        P(mutex);
        put();
        P(mutex);
        if(number&1){
            V(odd);
        }else{
            V(even);
        }
    }

P2:
    while(true){
        P(odd);
        P(mutex);
        getodd();
        V(mutex);
        V(empty);
        countodd();
    }

P3:
    while(true){
        P(even);
        P(mutex);
        geteven();
        V(mutex);
        V(empty);
        counteven();
    }
```

4.

```
Semaphore Smutex=1;          //对searchCnt的维护
Semaphore Se_De=1;           //搜索删除互斥
```

```
Semaphore In_In=1;          //插入插入互斥
Semaphore In_De=1;          //插入删除互斥
Semaphore De_De=1;          //删除删除互斥
int searchCnt=0;

Search:
    while(true){
        P(Smutex);
        if(search==0){
            P(Se_De);
        }
        searchCnt++;
        V(Smutex);
        search();
        P(Smutex);
        searchCnt--;
        if(search==0){
            V(Se_De);
        }
        V(Smutex);
    }

Insert:
    while(true){
        P(In_In);
        P(In_De);
        insert();
        V(In_De);
        V(In_In);
    }

Delete:
    while(true){
        P(Se_De);
        P(In_De);
        P(De_De);
        delete();
        V(De_De);
        V(In_De);
        V(Se_De);
    }
```