

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）。

```
int writecount = 0;
int readcount = 0;
semaphore RWmutex = 1; // reader & writer data访问互斥
semaphore readermutex = 1; // 互斥访问 readcount
semaphore writermutex = 1; // 互斥访问 writecount o'ra
semaphore Wmutex = 1; // writer data访问互斥，优先唤醒writer

function writer() {
    P(writermutex);
    writecount += 1;
    if(writecount == 1) {
        P(RWmutex);
    }
    P(Wmutex);
    V(writermutex);

    // write...

    P(writemutex)
    V(Wmutex);
    writecount -= 1;
    if(writecount == 0) {
        V(RWmutex);
    }
    V(writemutex);
}

function reader() {
    P(readermutex);
    readcount += 1;
    if(readcount == 1) {
        P(RWmutex);
    }
    V(readermutex);

    // read...

    P(readermutex);
```

```

    readcount -= 1;
    if(readcount == 0) {
        V(RWmutex);
    }
    V(readermutex);
}

```

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 eatingmutex = 1;
int eating = 0; // 在吃饭的人
int mustwait = 0; // 满5人时必须等待
semaphore waiting = 0;

function customer() {
    P(eatingmutex);
    if(eating == 5) {
        mustwait = 1;
        P(waiting);
    } else if(eating < 5) {
        eating += 1;
    }
    V(eatingmutex);

    // eating...

    P(eatingmutex);
    eating -= 1;
    if(mustwait && eating == 0) {
        V(waiting);
    }
    V(eatingmutex);
}

```

3. 进门问题。

(1) 请给出P、V操作和信号量的物理意义。

**P操纵 (Wait)**：尝试获取资源。当一个线程执行 P 操作时，其试图将信号量的值减少 1。如果信号量大于等于 0，则可以获取资源并继续执行；如果信号量为小于 0，则线程睡眠阻塞，知道资源可用时唤醒

**V操作 (Signal)：**释放资源。当线程执行 V 操作时，将信号量的值增加 1，表示释放一个资源。这样，其他线程如果在等待该资源，则可以被唤醒执行

**信号量：**一种同步计数器，用于控制多个线程/进程对共享资源的访问

(2) 一个软件公司有5名员工，每人刷卡上班。员工刷卡后需要等待，直到所有员工都刷卡后才能进入公司。为了避免拥挤，公司要求员工一个一个通过大门。所有员工都进入后，最后进入的员工负责关门。请用P、V操作实现员工之间的同步关系。

```
int entered = 0;
semaphore entered_m = 1;

int wait = 0;
semaphore wait_m = 1;

semaphore barrier = 0;

function employee() {
    P(wait_m);
    wait++;
    if(wait == 5) {
        V(barrier);
    }
    V(wait_m);

    P(barrier);
    V(barrier);

    P(entered_m);
    entered++;
    if(entered == 5) {
        // close the door...
    }
    V(entered_m);

    // work...
}
```

4. 搜索-插入-删除问题。三个线程对一个单链表进行并发的访问，分别进行搜索、插入和删除。搜索线程仅仅读取链表，因此多个搜索线程可以并发。插入线程把数据项插入到链表最后的位置；多个插入线程必须互斥防止同时执行插入操作。但是，一个插入线程可以和多个搜索线程并发执行。最后，删除线程可以从链表中任何一个位置删除数据。一次只能有一个删除线程执行；删除线程之间，删除线程和搜索线程，删除线程和插入线程都不能同时执行。

请编写三类线程的同步互斥代码，描述这种三路分类互斥问题。

```
semaphore c_m = 1; // 插入互斥
semaphore mutex = 1; // 资源访问

int sc_count = 0; // 插入和查询总量
semaphore sc_m = 1; // 插入和查询总量变量互斥

function query() {
    P(sc_m);
    sc_count++;
    if(sc_count == 1)
        P(mutex);
    V(sc_m);

    // query...

    P(sc_m);
    sc_count--;
    if(sc_count == 0)
        V(mutex);
    V(sc_m);
}

function insert() {
    P(c_m);

    P(sc_m);
    sc_count++;
    if(sc_count == 1)
        P(mutex);
    V(sc_m);

    // insert..

    P(sc_m);
    sc_count--;
    if(sc_count == 0)
        V(mute);
    V(sc_m);

    V(c_m);
}

function delete() {
```

```
P(mutex);  
  
// delete...  
  
V(mutex);  
}
```