Bounded Buffer Problem（Producer-Consumer problem）生产者消费者问题

There is a buffer of **n** slots and each slot is capable of storing one unit of data. There are **two processes** running, namely, **producer** and **consumer**, which are operating on the buffer.

缓冲区有 **n** 个插槽，每个插槽可存储一个单位的数据。有**两个进程**在缓冲区内运行，即**生产者和消费者。**

Producer希望能够向缓冲区中插入数据

Consumer希望从缓冲区中移除数据（前提是缓冲区中的数据量应该大于0）

要避免两个进程并发执行对缓存进行修改，因为可能导致并行操作使最终某个进程的修改结果丢失。

因此要引入semaphore（信号量）的概念，信号量可以被用来代表当前缓存中数据量的信息，例如在本题中empty代表buffer中有多少空着的位子，full代表buffer中有多少位子已经被占用了

```
// Producer 生产者
do
{
    // wait until empty > 0 and then decrement 'empty'
    // 持续等待直到empty的值 > 0，然后wait使empty的值-1
    wait(empty);
    // acquire lock
    // 获取互斥锁，在进入critical section的时候需要保证一次只能有一个进程进行操作
    wait(mutex);

    /* perform the insert operation in a slot */

    // release lock
    // 在当前进程操作完成之后释放掉互斥锁
    signal(mutex);
    // increment 'full'
    // 这里将会让full的值+1，代表生产者已经生产出来了一个新的东西占用了当前buffer中的一个空位
    signal(full);
} while(TRUE)
```

```
// Consumer 消费者
do
{
    // wait until full > 0 and then decrement 'full'
    // 等待当full > 0的时候，wait让full-1
    wait(full);
    // acquire the lock
    // 获取互斥锁
    wait(mutex);

    /* perform the remove operation in a slot */

    // release the lock
    // 释放互斥锁
    signal(mutex);
    // increment 'empty'
```

```
    // 消费者消费了一个东西，这时候可以用signal让empty+1
    signal(empty);
} while(TRUE);
```

Reader and Writer Problem 读者和写者问题

这里假设了两个类型的进程共同分享同一本书。并且规定可以用任意数量的读者进行阅读，但是每次只能最多有一名写者，而且必须是在没有读者进行阅读的情况下。

因此读者应该比写者的优先级更高，因此写者永远要等没有读者在读的时候。

引入一个信号量w和一个互斥锁m，一个整数变量记录当前正在读书的读者数量（初始化该变量为1），让w和m的初始值为1

```
// 写者 Writer
while(TRUE)
{
    // 在可以进行写操作的时候上锁，此时读者暂时无法上锁
    wait(w);

    /* perform the write operation */

    // 写完之后解锁
    signal(w);
}
```

```
// Reader 读者
while(TRUE)
{
    // 在有人读书的时候上m的锁，防止多个reader一起修改
    wait(m);
    read_count++;
    // 如果此时已经读者的数量不少于1，则要给写者上锁，防止有写者能够私自进行写操作
    if(read_count == 1)
        wait(w);

    //release lock
    // 解开读者的m锁
    signal(m);

    /* perform the reading operation */

    // acquire lock
    // 当当前进程的读者读完之后，再给m锁上
    wait(m);
    // 当前读书人数减一
    read_count--;
    // 如果此时读书的人数已经为0，则可以解开对写者的锁，代表此时如果有写者想写的话就可以进行写操作
    if(read_count == 0)
```

```
        signal(w);

    // release lock
    // 释放当前的读者的锁
    signal(m);
}
```

Dinning Philosophers Problem 哲学家进餐问题

用于评估需要将多个资源分配给多个进程的情况

假设有五位哲学家围坐在一张圆形餐桌旁。餐桌上有五根筷子每两个人中间放一个筷子，中间有一碗米饭。

每个哲学家要么是在吃饭要么是在思考，当需要吃饭的时候就需要拿起自己左手边和右手边的两个筷子，如果要思考就将自己的筷子放在左边和右边各一根。要考虑避免无穷引用。

一个由五个信号量组成的数组，stick[5]，代表五个筷子中的每一个。

哲学家必须在能同时获取左手和右手的筷子时才能吃饭

```
// 哲学家
while(TRUE)
{
    // i+1号哲学家要进餐的时候，需要将其左手边对应筷子的信号量-1
    wait(stick[i]);
    /*
    mod is used because if i=5, next
    chopstick is 1 (dining table is circular)
    */
    // 同时通过(i+1)%5可以算出他右手边对应的筷子的信号量-1
    wait(stick[(i+1) % 5]);

    /* eat */
    // 在吃完之后需要将他左手和右手边的信号量再+1，代表可以被他人使用
    signal(stick[i]);

    signal(stick[(i+1) % 5]);
    /* think */
}
```