读者写者问题

某男子足球俱乐部，有教练、队员若干。每次足球训练开始之前，教练、球员都需要先进入更衣室换衣服，可惜俱乐部只有一个更衣室。教练们脸皮薄，无法接受和别人共用更衣室。队员们脸皮厚，可以和其他队员一起使用更衣室。如果队员和教练都要使用更衣室，则应该让教练优先。请使用P、V操作描述上述过程的互斥与同步，并说明所用信号量及初值的含义。

> 【参考答案】本题中，教练就是写者，队员就是读者，不过是读者写者问题换了个马甲而已。按照题目要求，要求实现“写优先”。注意和王道书上的两种“读者-写者”解法对比学习。“写优先”代码如下：

```
semophore readLock=1;                //互斥信号量，用于给读者“上锁”
semophore writeLock=1;               //互斥信号量，用于给写者“上锁”
semophore rmutex=1;                 //互斥信号量，实现对readCount的互斥访问
semophore wmutex=1;                 //互斥信号量，实现对writeCount的互斥访问
int readCount=0, writeCount=0;  //读者、写者的数量

//读者进程（在这个题里就是可以多人一起共用更衣室的队员们）
Reader(){
while(true){
P(readLock);         //每个读者到达时先对 read 上锁
P(rmutex);
  readCount++;
  if(readCount==1) P(writeLock);  //第一个开始读的读者对写者上锁
V(rmutex);
V(readLock);         //每个读者正式开始读之前对 read 解锁
读者读...;
P(rmutex);
  readCount－－;
  if(readCount==0) V(writeLock);  //最后一个读完的读者对写者解锁
V(rmutex);
}
}

//写者进程（在这个题目里，对应必须独享更衣室的教练们）
Writer(){
while(true){
P(wmutex);
  writeCount++;
  if(writeCount==1) P(readLock);     //第一个到达的写者对读者上锁，这一步是实现“写优先”的关键
V(wmutex);
P(writeLock);    //每个写者开始写之前都要 P(write)，保证写者之间互斥，同时也能保证若当前有读者正在读，那么写者等待
写者写...;
V(writeLock);
P(wmutex);
    writeCount－－;
    if(writeCount==0) V(readLock);  //最后一个写者写完之后，对读者解锁
```

```
V(wmutex);
}
}
```

为了方便大家对比学习，下面再附上“读者优先”的实现，这种方式可能导致写者饥饿：

```
semaphore lock=1; //用于实现对共享文件的互斥访问
int count = 0;    //记录当前有几个读进程在访问文件
semaphore mutex = 1;  //用于保证对count变量的互斥访问

writer (){
while(1){
P(lock);    //写之前“加锁”
写文件…
V(lock);    //写完了“解锁”
}
}

reader (){
while(1){
P(mutex); //各读进程互斥访问count
if(count==0)
    P(lock);  //第一个读者，读之前“上锁”
count++;  //访问文件的读进程数+1
V(mutex);
读文件…
P(mutex); //各读进程互斥访问count
count--;  //访问文件的读进程数-1
if(count==0)  //由最后一个读进程负责
    V(lock);  //读完了“解锁”
V(mutex);
}
}
```

下面是“读写公平法”。也就是王道书里的第二种方法。

```
semaphore lock=1;          //用于实现对共享文件的互斥访问
int count = 0;           //记录当前有几个读进程在访问文件
semaphore mutex = 1;     //用于保证对count变量的互斥访问
semaphore queue = 1;     //用于实现“读写公平”。咸鱼注：可以将queue理解为一个“队列”，当资源暂不可访问时，无论读者、写者都需要公平排队

writer (){
while(1){
  P(queue); //先排队
  P(lock);    //尝试“上锁”
  V(queue); //唤醒下一个队头进程
  写文件…
```

```
  V(lock);     //使用完资源，解锁
}
}

reader (){
while(1){
  P(queue);      //先排队
  P(mutex);      //互斥访问count
  if(count==0)
      P(lock);     //第一个到达的读者尝试“上锁”
  count++;         //读者计数+1
  V(mutex);
  V(queue);      //唤醒队头进程
  读文件…
  P(mutex);      //互斥访问count
  count--;         //读者计数-1
  if(count==0)     //最后一个离开的读者，负责“解锁”
      V(lock);
  V(mutex);
}
}
```

对于上面三种解法，如果弄不清楚它们之间的区别，不妨带入一个例子看看。假设每个读者的读操作都耗时较长，读者写者到达的顺序是：

读者1——读者2——读者3——写者A——读者4——写者B——读者5

如果采用“读者优先”的实现方法，那情况是这样的：读者1到达并开始读，紧接着读者2、读者3到达，都可以开始读；写者A到达，暂时不能写；读者4到达，可以开始读；写者B到达，暂时不能写；读者5到达，可以直接开始读；等读者1、2、3、4、5都读完之后，写者A、写者B才可以依次进行写。

如果采用“读写公平法”的实现方法，那情况是这样的：读者1到达并开始读，紧接着读者2、读者3到达，都可以开始读；写者A到达，暂时不能写；读者4到达，暂时不能读；写者B到达，暂时不能写；读者5到达，暂时不能读；等读者1、2、3都读完之后，写者A开始写；写者A写完之后读者4开始读；读者4读完后写者B开始写；写者B写完后读者5开始读。

如果采用“写优先”的实现方法，那情况是这样的：读者1到达并开始读，紧接着读者2、读者3到达，都可以开始读；写者A到达，暂时不能写；读者4到达，暂时不能读；写者B到达，暂时不能写；读者5到达，暂时不能读；等读者1、2、3都读完之后，写者A开始写；写者A写完之后写者B开始写；写者B写完后读者4开始读，同时读者5也可以开始读。