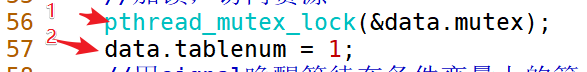
基础题：

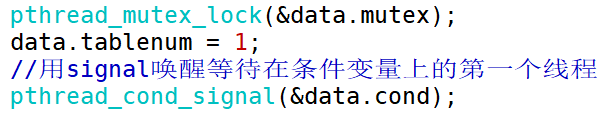
1. 理解线程的同步和互斥的概念，对锁的操作有哪些，对条件变量的使用有哪些函数？
   1. 线程互斥：当一个线程访问公共资源时，需要加锁防止另外的线程对他进行访问，实现资源的独占。
   2. 线程同步：
   3. 对锁：加锁、解锁、尝试加锁
   4. 条件变量函数：
      1. 初始化pthread\_cond\_init(&cond,NULL);
      2. 无条件等待 pthread\_cond\_wait(&cond,&mutex);
      3. 计时等待 pthread\_cond\_timedwait(&cond,&mutex,&timespec)；
      4. 条件变量的激活. pthread\_cond\_signal(&cond); pthread\_cond\_broadcast(&cond);
2. pthread\_cond\_wait函数为什么要传递一个mutex的指针？描述一下pthread\_cond\_wait的上半部和下半部都做了哪些事？

传指针的原因：

主函数中，要先加锁才能增加资源

而子线程没有资源时，也会进入wait等待，此时造成两个线程的死锁。

因此给wait传一把锁，让wait等待时可以先给锁解开，把自己挂起等待，

这个时候主线程就可以发signal信号了（signal被包裹再锁内，前面子线程解锁了这时主线程就可加锁进入signal模块），避免死锁。

所以wait的实现是先解锁，等待，然后尝试加锁。（子线程中Wait的解锁，可以让主线程成功完成加锁后解锁）

完整的wait上下半部：

一、 上半部 1. 在条件变量上面排队 2. 解锁 3. 睡眠

二、 下半部 1. 被唤醒

2. 加锁

1. 如果锁没有锁住，加锁成功

2. 如果锁是锁住的，线程会阻塞，等到加锁成功为止

3. 从pthread\_cond\_wait函数返回（线程从条件变量上被唤醒，不代表一定会立 刻从pthread\_cond\_wait 函数返回。）

---------------------------------------------------------------------

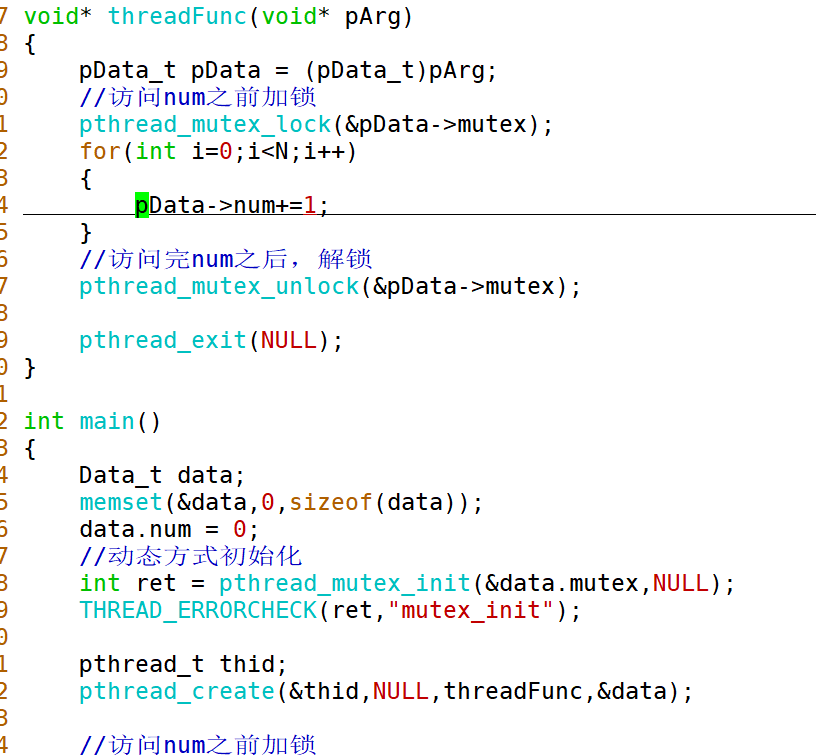
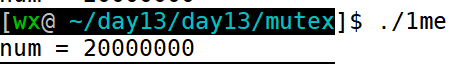
编程题：

1、 主线程定义一个整形数num=0，创建2个子线程，通过传参的方式把num传给子线程，让两个子线程分别对这个num各加1000万，

通过使用互斥锁，让两个子线程每次加完1000万后的结果都是2000万，注意使用清理函数，保证锁资源的释放。

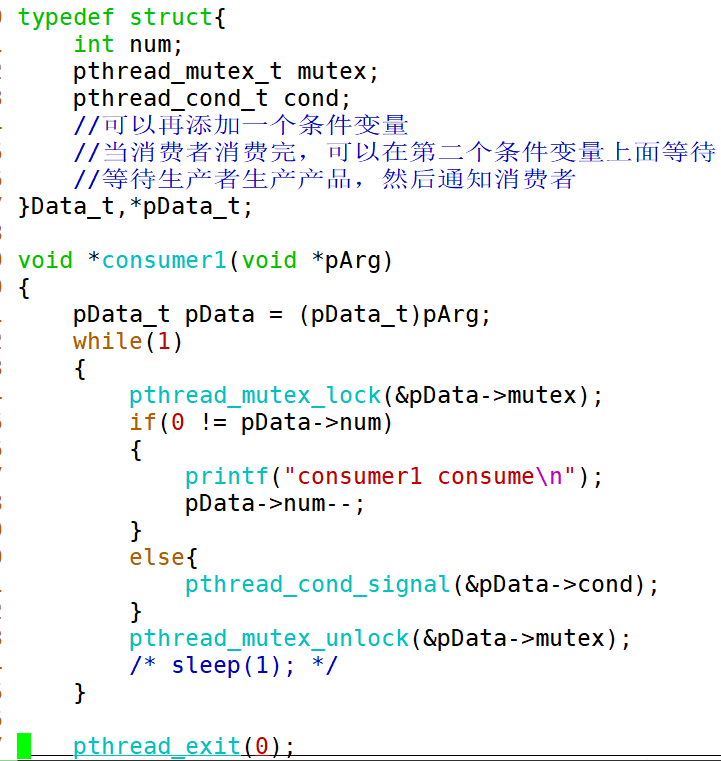
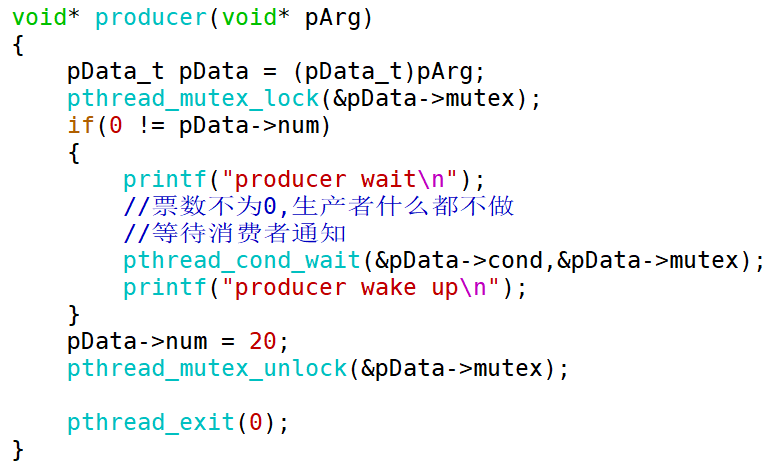
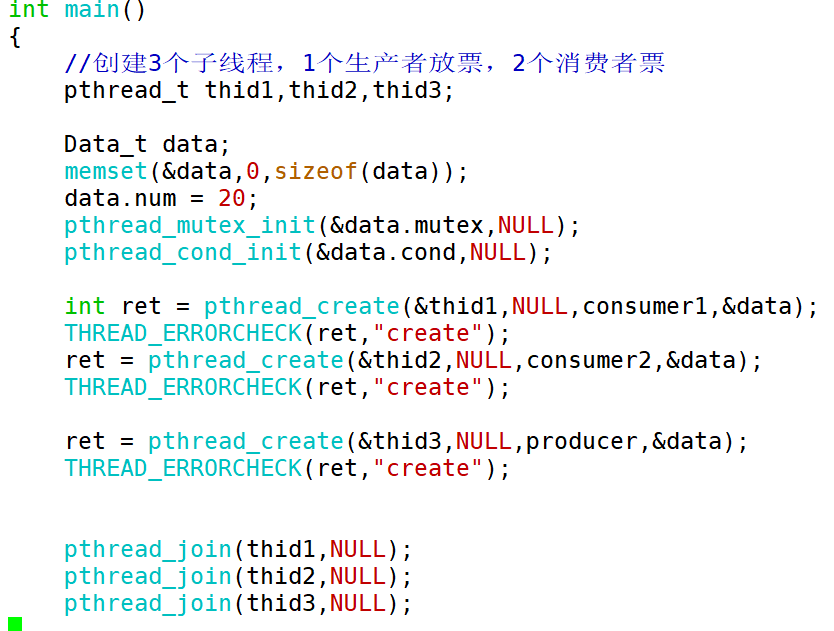
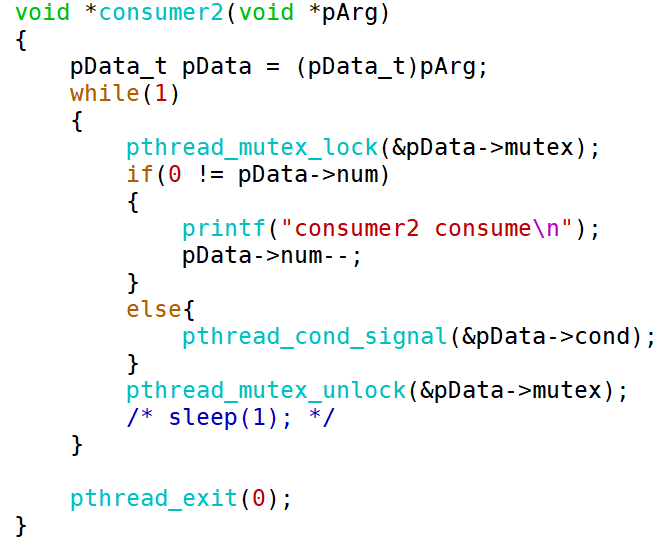
思考一下：

加锁和解锁应该放在for循环的外面还是for循环里面，为什么？

回答：加锁解锁应该在for循环外，因为加锁与解锁需要时间开销，若放在for循环里，则每次循环都会执行一遍“加锁解锁”，而时间片的划分通常可以满足很多次for循环，因此更符合执行多次for循环后之内来次加锁解锁的操作，这样程序更高效。 

2、编写一个生产者消费者卖票放票的程序，一个生产者，两个消费者，消费者有票就买票，没有票时等待通知，生产者运行时如果发现没票了，

就放票出来，然后通知消费者买票。

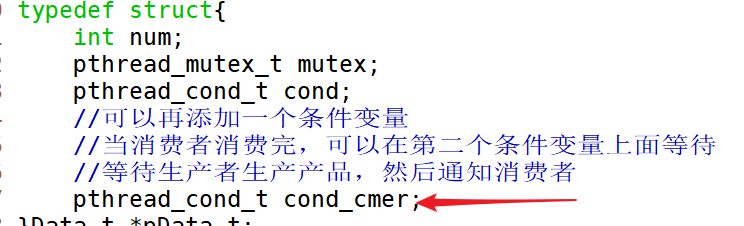
  

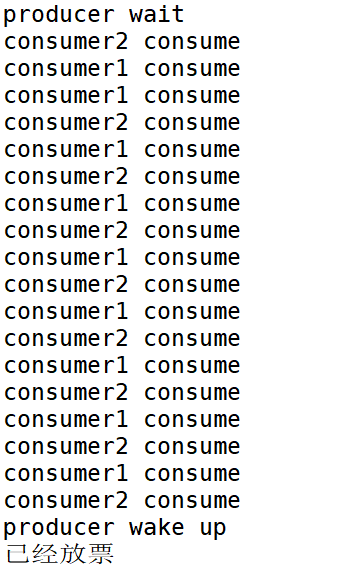
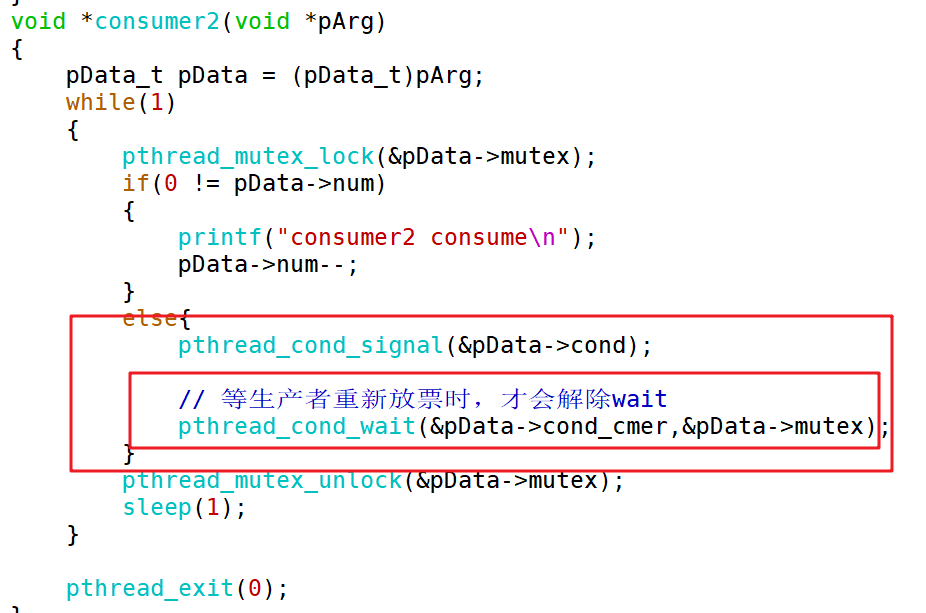
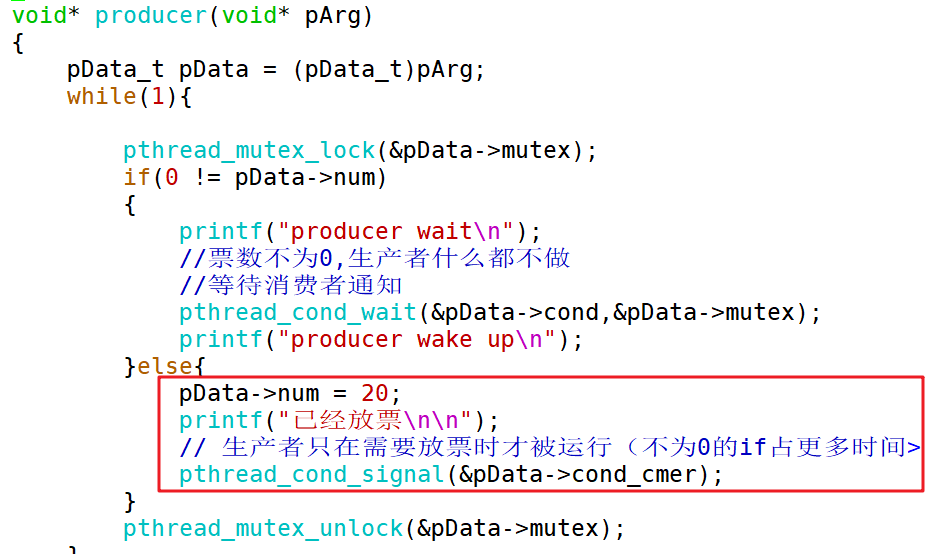
思考一下：能不能先判断余票的数量，再去加锁，如果这样写，会出现什么问题？

答：不能再锁的外面（先判断余票）判断余票，因为消费者进程都有对共享资源num写的操作，而此时生产者进程的读操作必须加锁进行，否则就可能会读取错误。

3、上面的模型有点不合理的地方，生产者需要一直运行，判断是否有余票，这样会跟消费者线程竞争CPU，比较浪费资源。把上面的程序改变一下，使用两个条件变量，

生产者运行后判断余票数量，如果余票为0，就放票并通知消费者，如果余票不为0，就等待消费者通知自己放票出来。





让生产者只在放票才运行signal（其他时间都只执行if中的wait，而wait可以做到几乎在睡眠）；

让消费者不消费时也wait（可以几乎在睡眠）

Wait（）可以减少cpu的消耗，在produce与consumer中都有了wait后，这样可以最大程度避免cpu的忙等

【资源是20】：并不代表执行20次if才会执行1次else，因为若没等到资源，else会连续执行，这样就可能执行次数远大于10。