**操 作 系 统**

**实 验 报 告**

**实验名称：实验4 同步互斥问题**

**姓名： 陈衍斌**

**学号： 16340042**

实验名称：同步互斥问题

一、实验目的：

1. 利用线程同步机制，实现生产者-消费者问题

2. 用信号量机制分别实现读者优先和写者优先的读者-写者问题

二、实验要求：

1. 见第6章教材236页的project。设计一个程序来解决有限缓冲问题，其中的生产者与消费者进程如图6.10 与图6.11 所示。

在6.6.1 小节中，使用了三个信号量: empty (以记录有多少空位)、full (以记录有多少满位)以及mutex (二进制信号量或互斥信号量，以保护对缓冲插入与删除的操作)。对于本项目， empty 与full 将采用标准计数信号量，而mutex 将采用二进制信号量。生产者与消费者作为独立线程，在empty、full、mutex 的同步前提下，对缓冲进行插入与删除。

本项目，可采用Pthread 。

2. 在Linux环境下，创建一个进程，此进程包含n个线程。用这n个线程来表示n个读者或写者。每个线程按相应测试数据文件(后面有介绍)的要求进行读写操作。用信号量机制分别实现读者优先和写者优先的读者-写者问题。

读者-写者问题的读写操作限制(仅读者优先或写者优先)：

1)写-写互斥，即不能有两个写者同时进行写操作。

2)读-写互斥，即不能同时有一个线程在读，而另一个线程在写。

3)读-读允许，即可以有一个或多个读者在读。

读者优先的附加限制：如果一个读者申请进行读操作时已有另一个读者正在进行读操作，则该读者可直接开始读操作。

写者优先的附加限制：如果一个读者申请进行读操作时已有另一写者在等待访问共享资源，则该读者必须等到没有写者处于等待状态后才能开始读操作。

运行结果显示要求：要求在每个线程创建、发出读写操作申请、开始读写操作和结束读写操作时分别显示一行提示信息，以确定所有处理都遵守相应的读写操作限制。

三、实验过程：

**1．**首先要明确测试数据的文件格式：测试数据文件包括n行测试数据，分别描述创建的n个线程是生产者还是消费者，以及生产者或消费者存放或取产品的开始时间和持续时间。每行测试数据包括四个字段，各个字段间用空格分隔。第一字段为一个正整数，表示线程序号。第二字段表示相应线程角色，P表示生产者，C表示消费者。第三字段为一个正数，表示存放或取出操作的开始时间：线程创建后，延迟相应时间(单位为秒)后发出对共享资源的使用申请。第四字段为一个正数，表示操作的持续时间。第五字段为一个正数（仅生产者有），表示生产的产品号。当线程申请成功后，开始对共享资源的操作，该操作持续相应时间后结束，并释放共享资源。

下面是一个测试数据文件的例子：

1 C 3 5

2 P 4 5 1

3 C 5 2

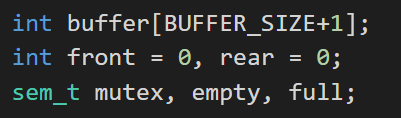
4 C 6 5

5 P 7 3 2

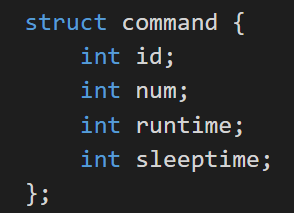
6 P 8 4 3

注意：在创建数据文件时，由于涉及到文件格式问题，最好在记事本中手工逐个键入数据，而不要拷贝粘贴数据，否则，本示例程序运行时可能会出现不可预知的错误。

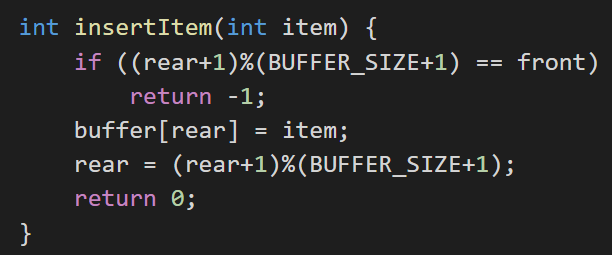
首先声明全局变量，设置缓冲区，存放生产者存放的产品，并设置头尾指针。

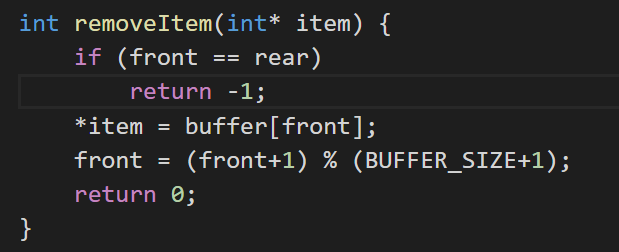


然后定义一个用于存放测试数据的结构体：



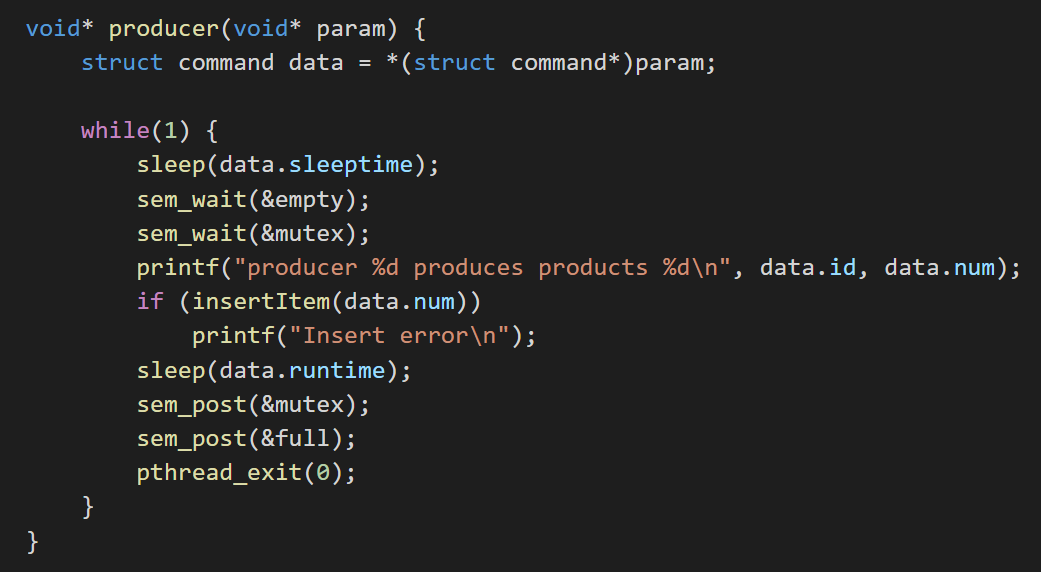
书本上也提到了，缓冲区可以通过如下两个函数实现：insertItem()和removeItem()。这两个函数将为生产者和消费者线程所分别使用。



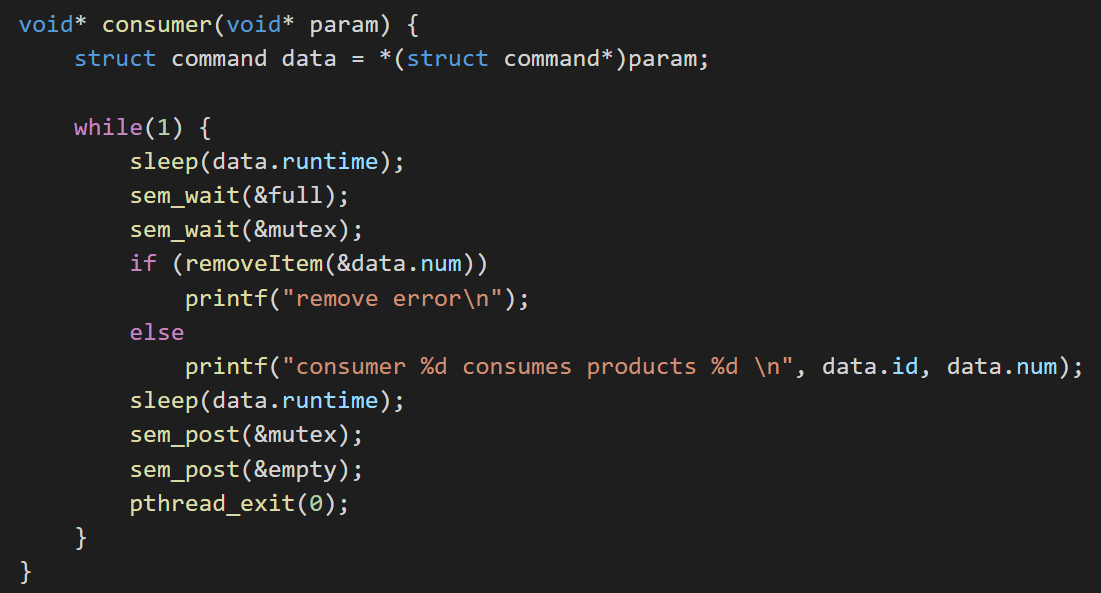


接下来1将传入的参数param转化为结构体，

写生产者的线程：

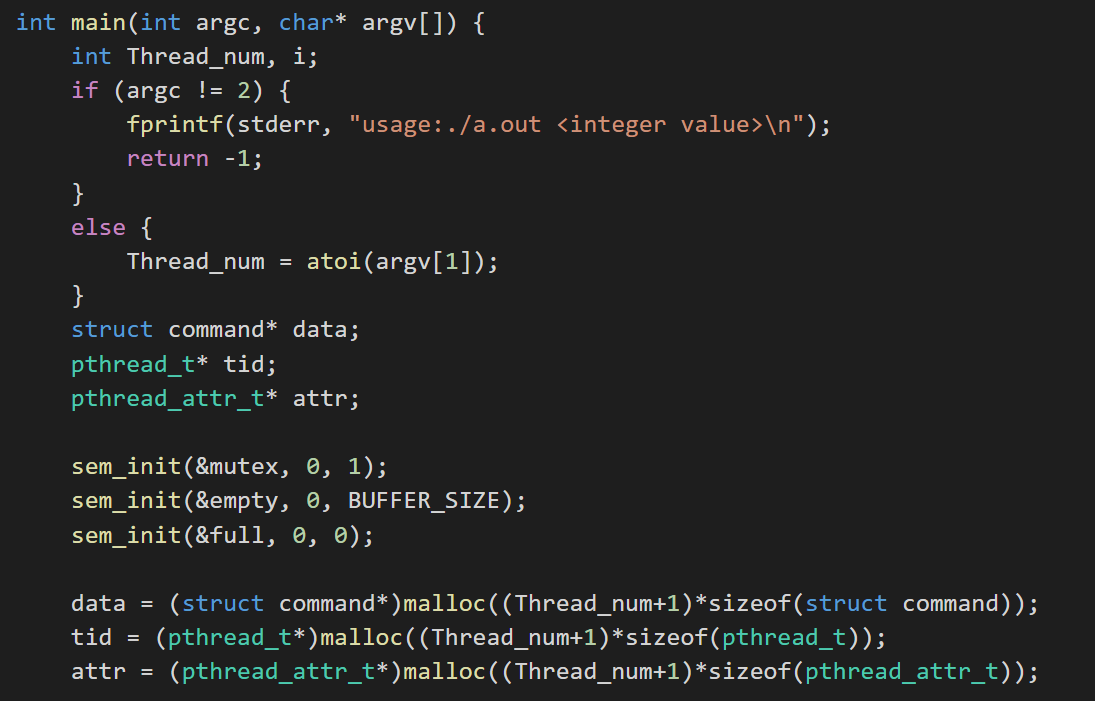


然后是消费者的线程：

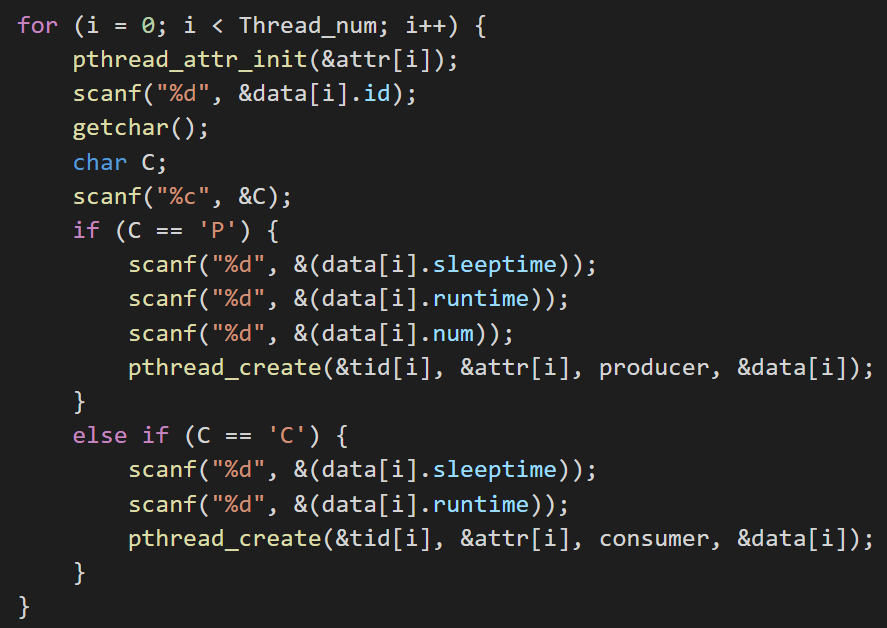


在各个线程创建好以后，就可以开始主函数的编写了。当程序运行时，传入argv[1]表示生产者和消费者的总数，即生产者和消费者线程总数。

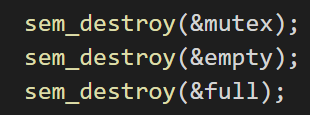
然后初始化各个信号量。结构体data存放所有的生产者和消费者信息。



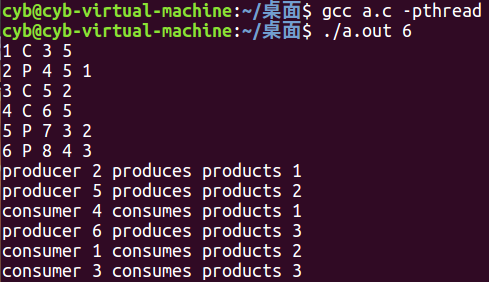
接下来生成生产者线程或生成消费者线程：



最后释放信号量mutex, empty, full。



最后编译程序，运行的结果如下：



**2．**先明确测试数据文件格式：测试数据文件包括n行测试数据，分别描述创建的n个线程是读者还是写者，以及读写操作的开始时间和持续时间。每行测试数据包括四个字段，各个字段间用空格分隔。第一字段为一个正整数，表示线程序号。第二字段表示相应线程角色，R表示读者，W表示写者。第三字段为一个正数，表示读写操作的开始时间：线程创建后，延迟相应时间(单位为秒)后发出对共享资源的读写申请。第四字段为一个正数，表示读写操作的持续时间。当线程读写申请成功后，开始对共享资源的读写操作，该操作持续相应时间后结束，并释放共享资源。

下面是一个测试数据文件的例子：

1 R 3 5

2 W 4 5

3 R 5 2

4 R 6 5

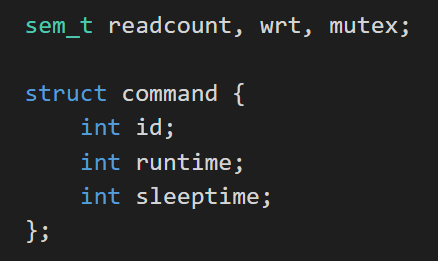
5 W 7 3

注意：在创建数据文件时，由于涉及到文件格式问题，最好在记事本中手工逐个键入数据，而不要拷贝粘贴数据，否则，本示例程序运行时可能会出现不可预知的错误。

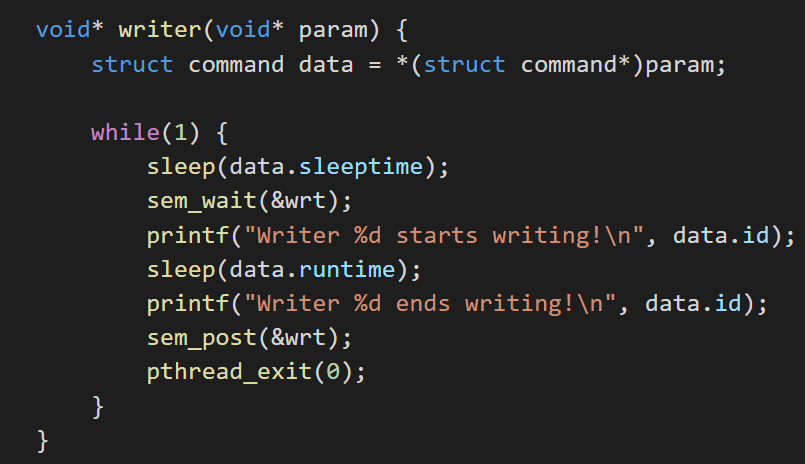
整体思路：可以将所有读者和所有写者分别存于一个读者等待队列和一个写者等待队列中，每当读允许时，就从读者队列中释放一个或多个读者线程进行读操作；每当写允许时，就从写者队列中释放一个写者进行写操作。

(1) 读者优先：读者优先指的是除非有写者在写文件，否则读者不需要等待。所以可以用一个整型变量read\_count记录当前的读者数目，用于确定是否需要释放正在等待的写者线程(当read\_count=0时，表明所有的读者读完，需要释放写者等待队列中的一个写者)。每一个读者开始读文件时，必须修改read\_count变量。因此需要一个互斥对象mutex来实现对全局变量read\_count修改时的互斥。另外，为了实现写-写互斥，需要增加一个临界区对象write。当写者发出写请求时，必须申请临界区对象的所有权。通过这种方法，也可以实现读-写互斥，当read\_count=1时(即第一个读者到来时)，读者线程也必须申请临界区对象的所有权。当读者拥有临界区的所有权时，写者阻塞在临界区对象write上。当写者拥有临界区的所有权时，第一个读者判断完“read\_count==1”后阻塞在write上，其余的读者由于等待对read\_count的判断，阻塞在mutex上。

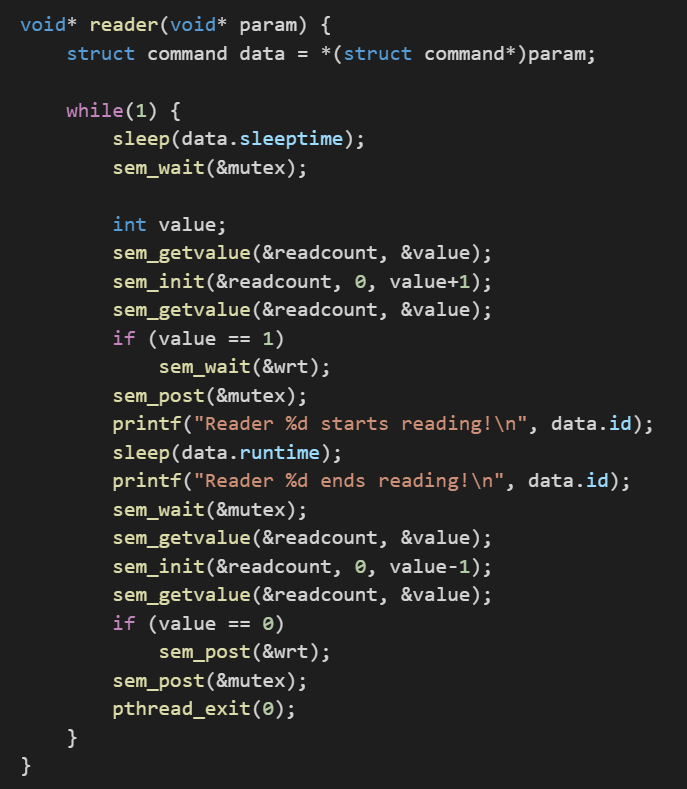
先定义各个信号量，然后定义用于存放测试数据的结构体：



然后定义写者线程：

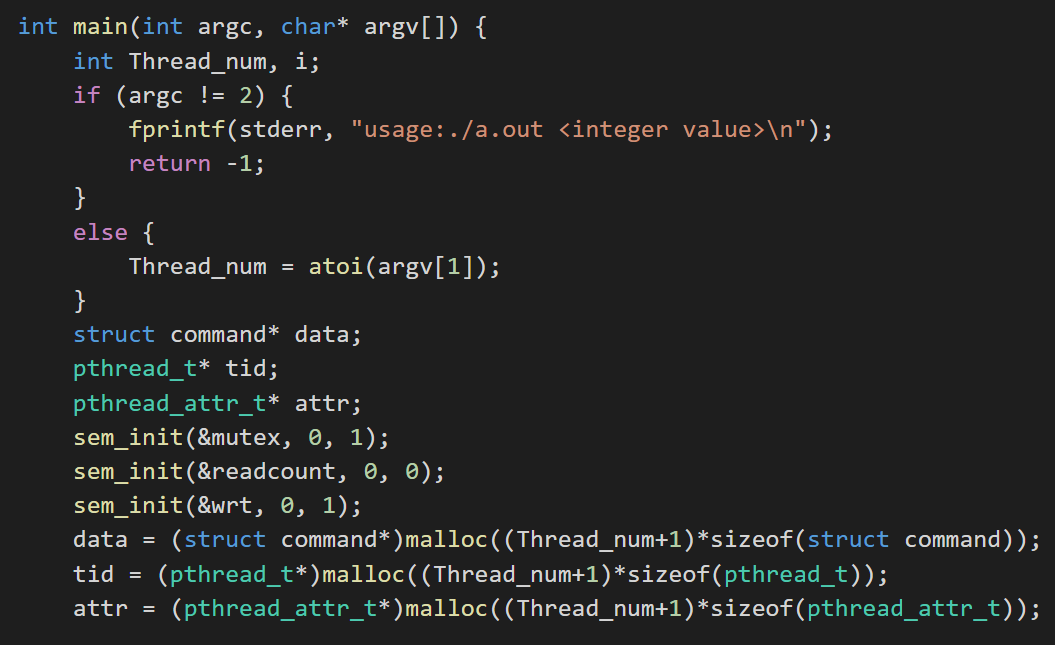


读者线程：



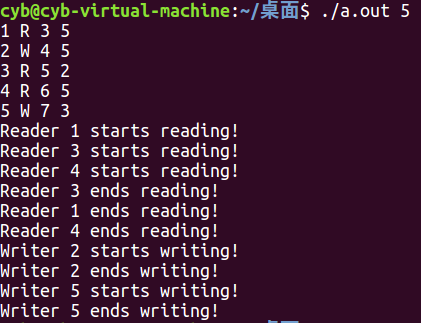
Main函数：传入argv[1]是线程的总数量，表示读者和写者的总数量。

其他的顺着写即可：





运行的结果如下：



可以看到，读者的读操作结束后，写者才开始写操作，这说明是读者优先。

(2) 写者优先：写者优先与读者优先类似。不同之处在于一旦一个写者到来，它应该尽快对文件进行写操作，如果有一个写者在等待，则新到来的读者不允许进行读操作。为此应当添加一个整型变量write\_count，用于记录正在等待的写者的数目，当write\_count=0时，才可以释放等待的读者线程队列。

为了对全局变量write\_count实现互斥，必须增加一个互斥对象mutex2。

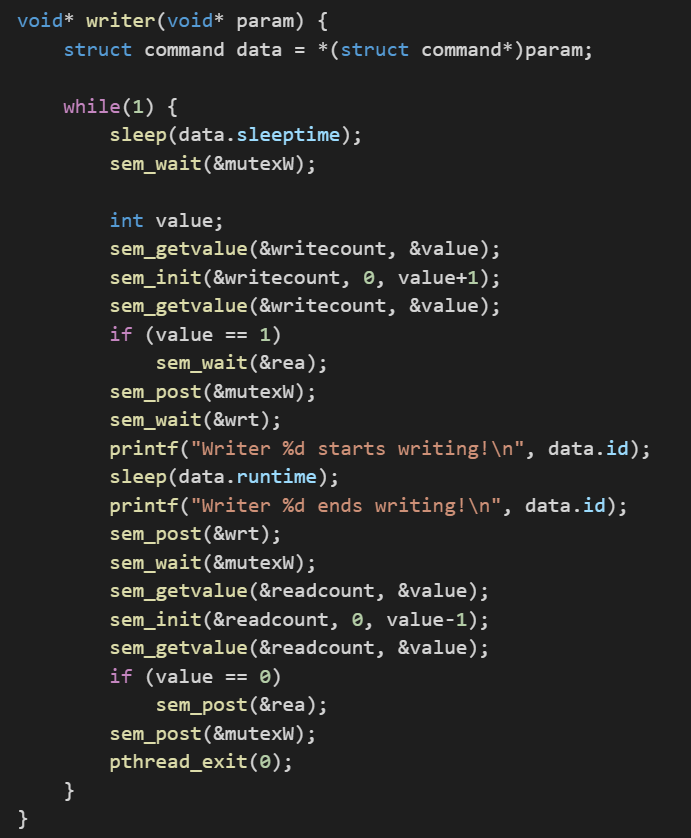
为了实现写者优先，应当添加一个临界区对象read，当有写者在写文件或等待时，读者必须阻塞在read上。同样，有读者读时，写者必须等待。于是，必须有一个互斥对象RW\_mutex来实现这个互斥。

有写者在写时，写者必须等待。

读者线程要对全局变量read\_count实现操作上的互斥，必须有一个互斥对象命名为mutex1。

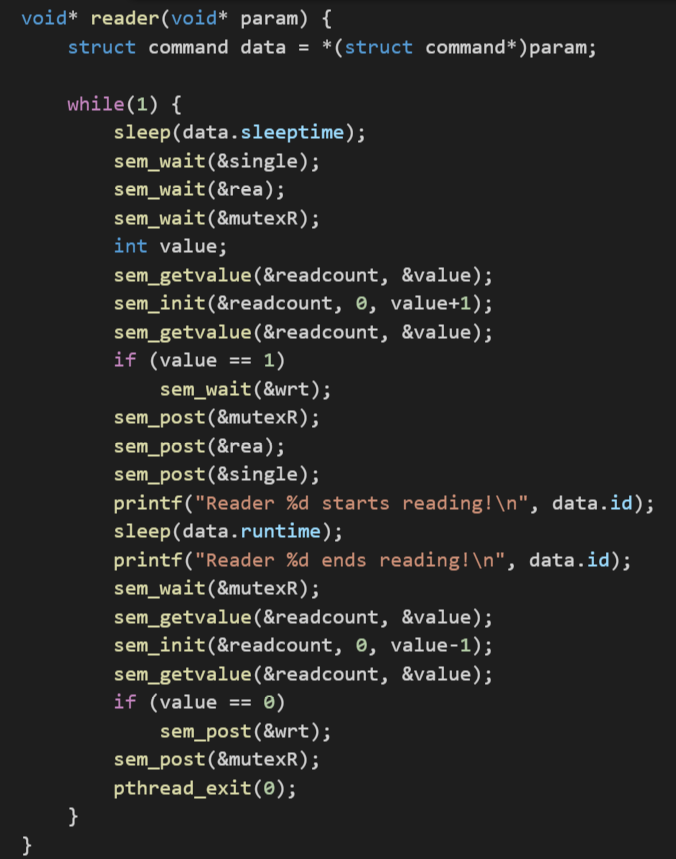
主要是需要修改读者和写者的进程，main函数的总体思路不变。

写者的进程如下：



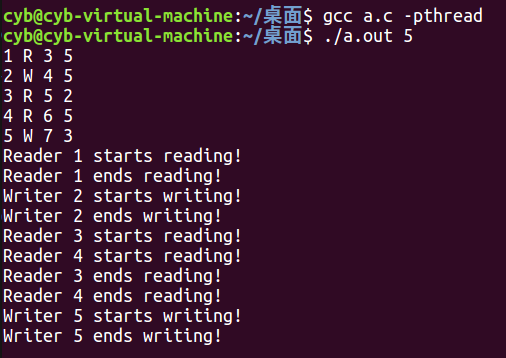
其中single用来保证每次只有一个读者进入临界区。

读者的进程如下：



Main函数区别不大，就不贴了。

编译后运行结果如下：



可以看到，中间先进行了写操作，然后停止了写操作，然后才进行读操作。所以说明是写者优先。

总结：总体来说，这次实验相比于之前，要难了不少。不过老师的ppt上有了不少提示，而且还给了数据输入以及变量的定义，这给了我不少启发。通过本次实验，我学习了关于多线程之间的互斥和同步问题，受益匪浅。