**操 作 系 统**

**实 验 报 告**

**实验名称：实验四 同步互斥问题**

**姓名： 王迎旭**

**学号： 16340226**

实验名称：**同步互斥问题**

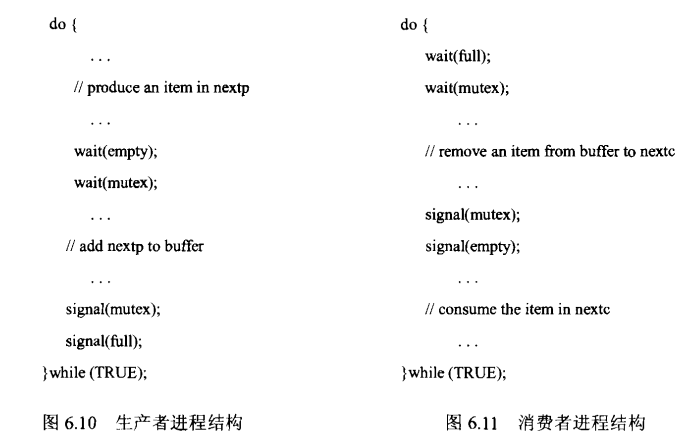
一、实验目的：

1. 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. 生产者与消费者问题

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。

1. 读者与写者问题

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。

1. 实验过程：

**Ⅰ、生产者与消费者问题**

（1）明确目标程序的需求：

用线程同步机制，实现生产者-消费者问题

（2）搜索题目所涉及相关资料：

1. Mutex
2. Mutex的三种类型

① 快速mutex:一个线程必须在未锁时获得mutex，否则必须等待。

② 递归mutex:拥有mutex的线程可多次加锁而不必等待，但此时其他线程必须等待。

③ 错误检测mutex:如果mutex已被锁，其他进程使用pthread

\_mutex\_lock()加锁时，将返回 EDEADLK

2）Mutex类型选择

通过初始化参数选择类型。POSIX线程库提供了更快速和直接初始化方法 :

① pthread\_mutex\_t fastmutex =

PTHREAD\_MUTEX\_INITIALIZER;//快速mutex

② pthread\_mutex\_t recmutex =

PTHREAD\_RECURSIVE\_MUTEX\_INITIALIZER\_NP; //递归mutex

③ pthread\_mutex\_t errchkmutex =

PTHREAD\_ERRORCECK\_MUTEX\_INITIALIZER\_NP; //错误检测mutex

3）Mutex的初始化

线程的互斥量数据类型是pthread\_mutex\_t。 使用前要对它进行初始化，有两种方法:

① 静态分配的互斥量的初始化：

static pthread\_mutex\_t mutex =

PTHREAD\_MUTEX\_INITIALIZER

② 动态分配的互斥量的初始化：

A、首先申请内存(malloc)，

B、再用pthread\_mutex\_init进行初始化。

C、释放时，必须先调用pthread\_mutex\_destroy，而后再释放内存(free)。

4）可对Mutex进行操作的函数

① int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t \*mutexattr);//初始化mutex.

② int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);//对mutex的加锁，不成等待

③ int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);//尝试加锁，不成返回

④ int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);//解锁

⑤ int pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex);//销毁mutex

5）线程对Mutex操作的性质

① 原子性。对mutex的加锁和解锁操作是原子的，一个线程进行 mutex 操作的过程中，其他线程不能对同一个 mutex 进行其他操作。

② 单一性。拥有mutex的线程除非释放mutex，否则其他线程不能拥有此mutex。

③ 非忙等待。等待mutex的线程处于等待状态，直到要等待的mutex处于未加锁状态，这时操作系统负责唤醒等待此mutex的线程。

1. POSIX信号量

1）POSIX信号量性质

① POSIX 信号量在多线程编程中可以起到同步或互斥的作用。用POSIX 信号量可以实现传统操作系统P、V操作(即对应课本的wait、signal)。

② 由于POSIX信号量不是内核负责维护，所以当进程退出后，POSIX信号量自动消亡。

1. POSIX信号量的含义

① sem\_init()原型：int sem\_init(sem\_t \*sem, int pshared, unsigned int value);

作用：创建信号量。sem为指向信号量结构的一个指针；pshared不为０时此信号量在进程间共享，否则只能为当前进程的所有线程共享；value给出了信号量的初始值。

② sem\_post()

作用：相当于signal操作，释放资源。用来增加信号量的值。当有线程阻塞在这个信号量上时，调用这个函数会使其中的一个线程被唤醒。

③ sem\_wait()：

作用：相当于wait操作，即申请资源。这是一个阻塞的函数。测试所指定信号量的值,它的操作是原子的。若sem>0，那么它减1并立即返回。若sem==0，则睡眠直到sem>0，此时立即减1，然后返回。

④ sem\_trywait ()

作用：是函数sem\_wait()的非阻塞版本。若信号量大于0，它直接将信号量sem的值减一并返回0；否则，它立即返回错误类型EAGAIN。

⑤ sem\_getvalue()

作用：获得信号量当前值

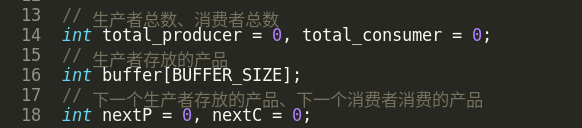
⑥ sem\_destroy()

作用：用来释放信号量。

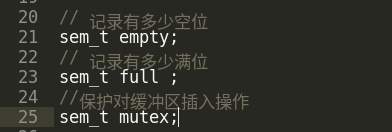
（3）消费者生产者程序设计

思路梳理（截图+文字叙述）

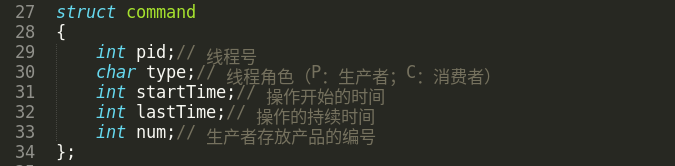
① 创建全局变量用于处理消费者与生产者的编号与其数目统计问题



② 创建POSIX信号量用于处理互斥问题



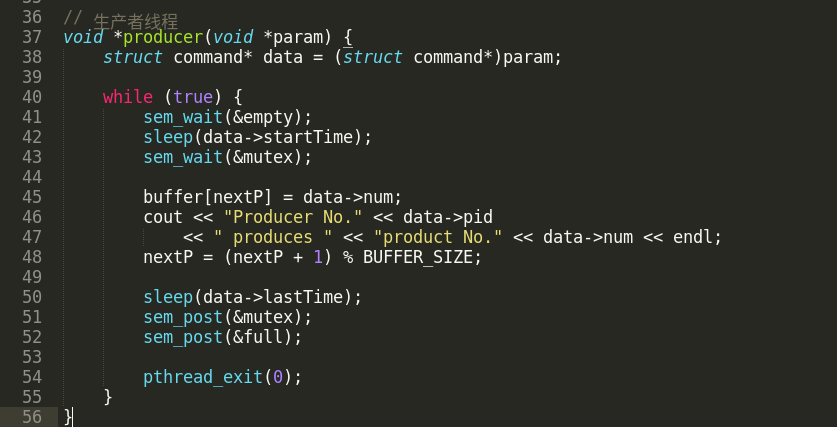
③ 创建command结构体按顺序存储PPT中所给的指令



④ 创建生产者线程函数

1. 将传入参数param转化为一个command的结构体，包含生产者的pid（线程号），type（即是P，线程角色：生产者），startTime（开始时间），lastTime（持续时间），num（生产者所生成的产品编号）。
2. 进入while循环：
3. 开始阻塞信号empty，等待缓冲区是否有空位；
4. 睡眠sleep开始时间startTime，阻塞mutex信号，修改缓冲区，把生成的编号为data->num的产品放入缓冲区，打印出已完成生产的信息，修改缓冲区指针。
5. 睡眠sleep持续时间lastTime，释放信号量mutex和full，允许消费者对缓冲区进行修改和取出产品。

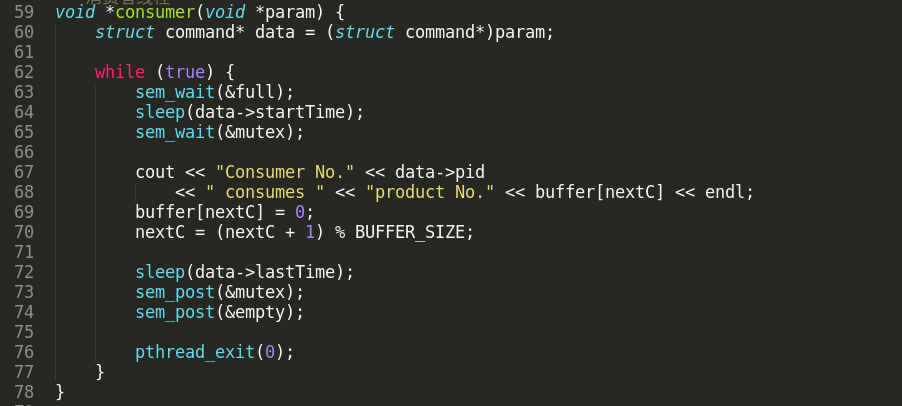
3）退出循环。



⑤ 消费者线程函数

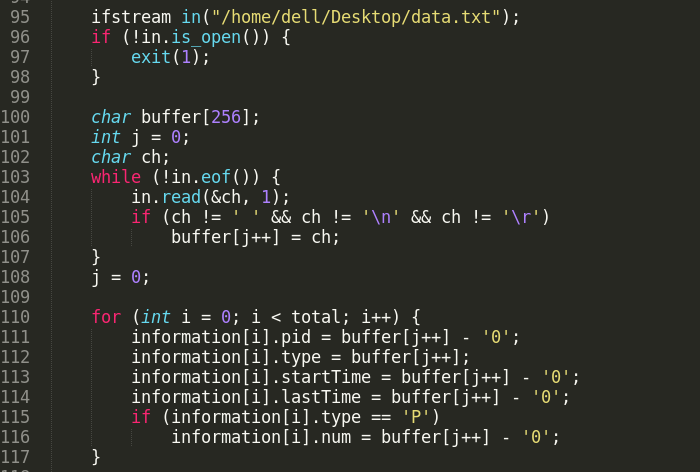
1. 将传入参数param转化为一个command的结构体（注意此时消费者没有num），包含生产者的pid（线程号），type（即是C，线程角色：消费者），startTime（开始时间），lastTime（持续时间）。
2. 进入while循环：
3. 开始阻塞信号full，等待缓冲区是否有产品；
4. 睡眠sleep开始时间startTime，阻塞mutex信号，修改缓冲区，取出消费编号为buffer[nextC]的产品，打印出已完成消费的信息，修改缓冲区指针，缓冲区减少一个产品；
5. 睡眠sleep持续时间lastTime，释放mutex和empty，允许生产者对缓冲区进行修改和向其中存放产品。

3）退出循环。

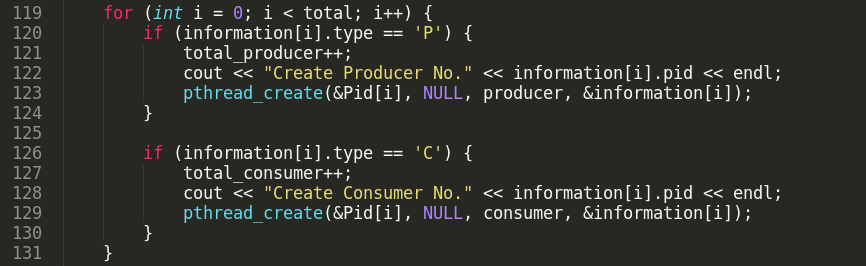


⑥ 创建ifstream读写流用于读取txt文件中的指令数据，同时完成对 指令的解码并按顺序赋值给指令结构体。

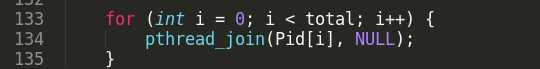
这里由于所给的指令中有4位的也有5位的，所以就选择了用数据流进行操作，来保证指令读取的正确性。



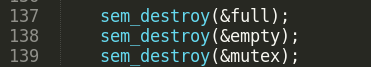
⑦ 根据指令集的指示，完成线程的创建并为每个线程分配对应的行为。



⑧ 非阻塞式调用线程

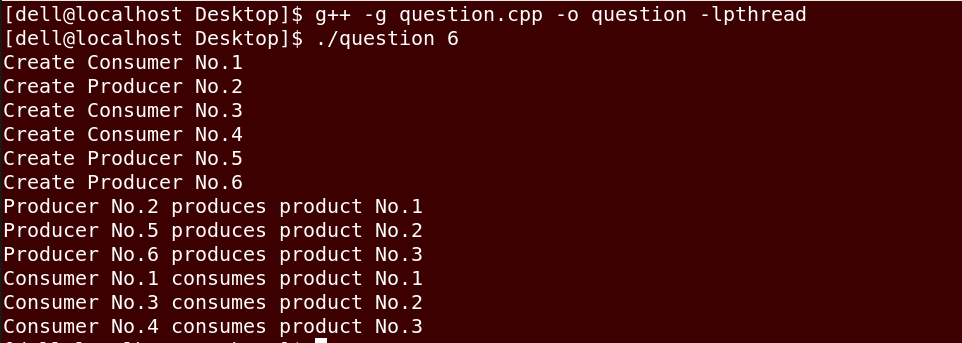


⑨ 删除信号量，终止程序



（4）程序测试与结果分析

程序测试：



结果分析：

从运行结果上我们可以看到，三个生产者进程完成自己的任务之后，三个消费者才开始自己的消费过程，但是为什么是如此？我们不妨从指令集上一条一条来分析。

指令集的指令如下：

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

① 第一条指令，消费者线程请求访问，但是这时候由于**full == 0**，所以这个线程就被阻塞在wait（full）这里，无法继续往下运行。

② 第二条指令，生产者线程请求访问，这时候由于程序开始时候empty被初始化为5，所以可以越过wait（empty）这一步，empty - 1后往下走，随后等待4毫秒，在进行到wait（mutex）这一步时候，由于mutex是1，所以可以越过wait（mutex）这一步，mutex - 1后继续往下走，这时候由于信号量mutex = 0 ，所以其他线程也就无权访问共享变量，也就保证了互斥性。往下走的过程中，生产了编号为1的产品，**第一个输出提示信息**。随后等待5毫秒，发出信号post（mutex），此时mutex -> 1，接着post（full），full ++ ,这一步后也就保证了消费者可以进行消费。

③ 第三条指令第四条指令，仍然是和第一条一样卡在wait（full）这里；这两条指令卡在这里是因为，第二条指令尚未完成时候，第三条第四条指令已经在wait（full）了，full仍然是0，所以就被阻塞。

④ 第五条指令，随着第二条指令的post（mutex）完成，第五条指令开始进入像第二条指令一样的运行过程，所以**第二个输出提示信息**。

⑤ 第六条指令，由于线程调用的顺序是先No.5后No.6所以，生产者5的生产过程早于生产者6。随后第五条指令完成任务后，紧接着就把访问权交给了第六条指令，进而完成创建任务，所以**第三个输出提示信息**。

⑥ 第六条指令完成后，当mutex重新被设置为1后，第一条指令率先夺取到访问权，于是就完成消费过程，所以**第四个输出提示信息**；剩下的第三条第四条指令过程与此类似，都是按照被创建的先后顺序完成信息输出。

**Ⅱ、读者与写者问题**

Part 1 读者优先

（1）明确目标程序的需求

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。

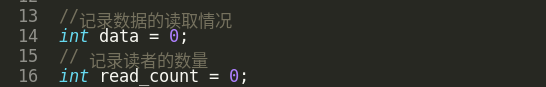
另外，为了实现写-写互斥，需要增加一个临界区对象write。当写者发出写请求时，必须申请临界区对象的所有权。通过这种方法，也可以实现读-写互斥，当read\_count=1时(即第一个读者到来时)，读者线程也必须申请临界区对象的所有权。

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。

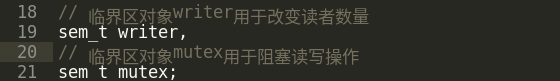
（2）读者优先程序设计

思路梳理（截图+文字叙述）

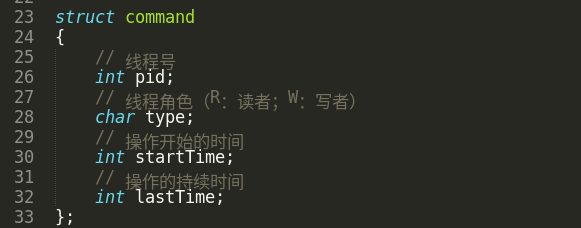
① 创建全局变量用于记录读者的数目以及读取的内容



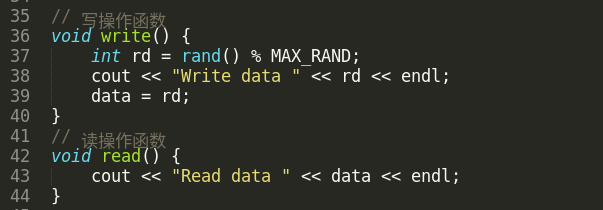
② 创建POSIX信号量用于处理互斥问题



③ 创建command结构体按顺序存储PPT中所给的指令



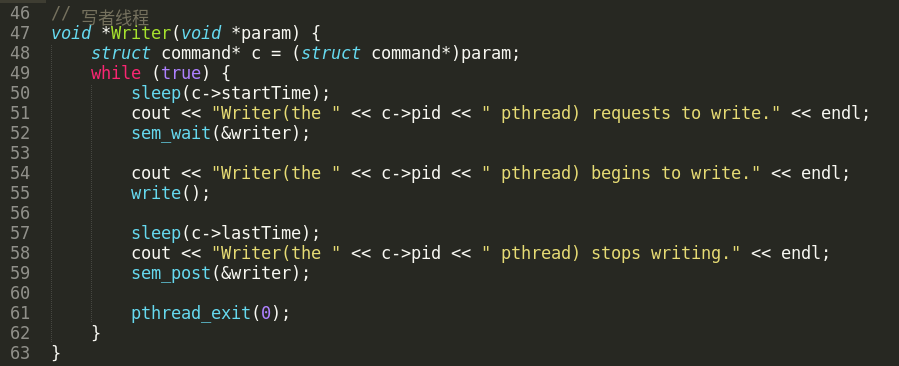
④ 创建读写函数用于显示此过程中处理的数据



⑤ 创建写者线程函数：

1. 将传入参数param转化为一个command的结构体，包含写者的 pid（线程号），type（即是W，线程角色：写者），startTime（开 始时间），lastTime（持续时间）。

1. 进入while循环
2. 睡眠sleep开始时间startTime；
3. 打印出线程号为pid的线程正在申请资源信息，并开始 阻塞writer；
4. 打印出开始写的信息，并执行写操作函数write()，显 示写入的信息；
5. 睡眠sleep持续时间lastTime； 打印出结束写的信息 并释放资源，解除阻塞信号writer；
6. 退出循环。



⑥ 创建读者线程函数：

1）将传入参数param转化为一个command的结构体，包含读者的pid（线程号），type（即是R，线程角色：读者），startTime（开始时间），lastTime（持续时间）。

2）进入while循环

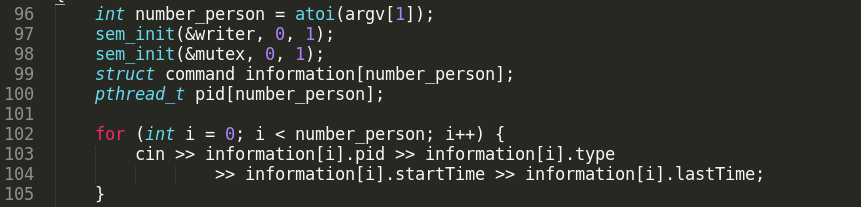
1. 睡眠sleep开始时间startTime；
2. 打印出线程号为pid的线程正在申请资源信息，并开始阻塞互斥信号mutex； 读者数量read\_count加1，检查read\_count，如果为1，那么阻塞writer。判断后释放互斥信号mutex。
3. 打印出开始读取的信息并执行读函数read(); 睡眠sleep持续时间lastTime；
4. 打印出结束写的信息并释放资源，并阻塞互斥信号mutex； 读者数量read\_count减1，检查read\_count，如果为0，那么释放信号writer。判断后解除互斥信号mutex。

3）退出循环

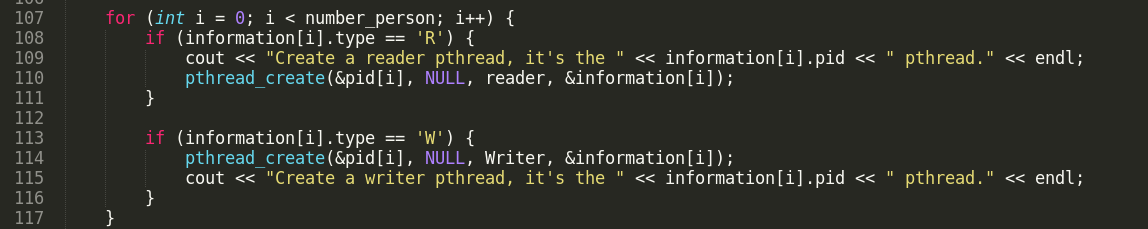


⑦ 进入main函数：

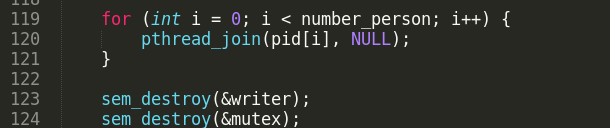
1. 创建信号量并完成初始化
2. 创建线程结构体
3. 完成指令的输入



⑧ 顺序遍历指令集完成线程的创建：

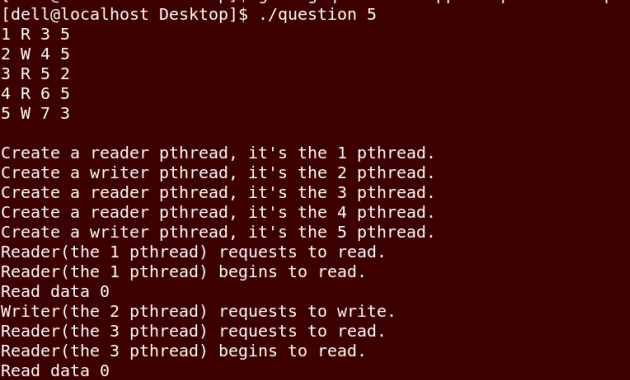


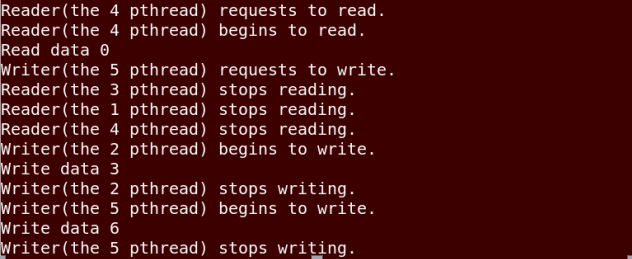
⑨ 非阻塞调用线程随后销毁信号量结束程序：



（3）程序测试与结果分析

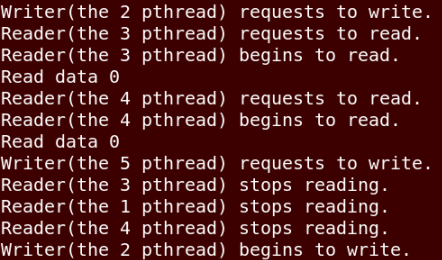
程序测试：+





结果分析：

读者优先就意味着当有读者写者同时请求进行线程操作时候，读者要先进行操作，所以程序的运行结果中



这一段：读者线程2与写者线程3同时请求操作，但是读者线程 2先进行操作，同时这个过程中也有个读者线程4请求操作，当这两个 线程读操作完成之后，发出了stop指令之后，写者线程2才开始进行 写操作，这也就体现了读者优先的操作，也就说明了程序的正确性。

Part 2 写者优先

（1）明确目标程序的需求

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。

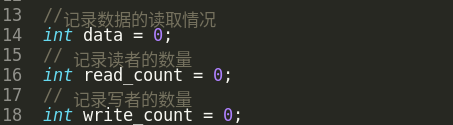
为了实现写者优先，应当添加一个临界区对象read，当有写者在写文件或等待时，读者必须阻塞在read上。同样，有读者读时，写者必须等待。于是，必须有一个互斥对象RW\_mutex来实现这个互斥。

有写者在写时，写者必须等待。

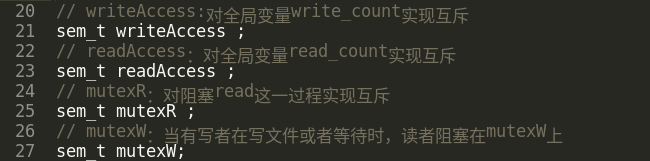
读者线程要对全局变量read\_count实现操作上的互斥，必须有一个互斥对象命名为mutex1。

（2）写者优先程序设计

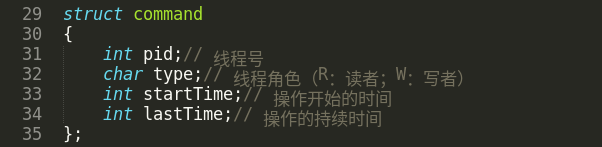
① 创建全局变量用于记录读者写者的数目以及读取的内容



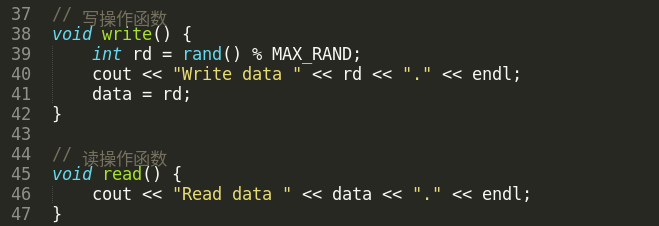
② 创建POSIX信号量用于处理互斥问题



③ 创建command结构体按顺序存储PPT中所给的指令



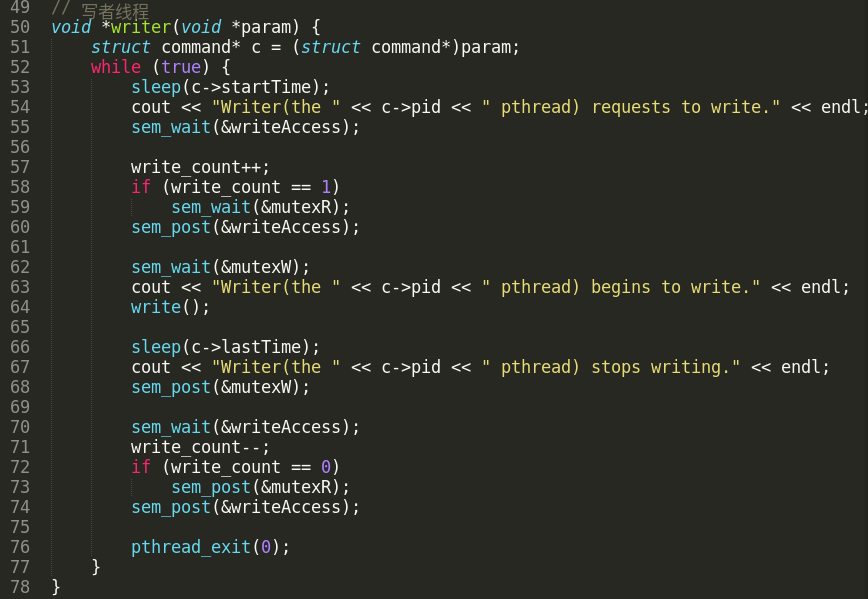
④ 创建读写函数用于显示此过程中处理的数据



⑤ 创建写者线程函数：

1. 将传入参数param转化为一个command的结构体，包含写者的pid（线程号），type（即是W，线程角色：写者），startTime（开始时间），lastTime（持续时间）。
2. 进入while循环：

1. 睡眠sleep开始时间startTime； 打印出线程号为pid的线程正在申请资源信息，并开始阻塞writerAccess；
2. 写者数量write\_count加1，检查write\_count，如果为1，那么阻塞mutexR。判断后释放互斥信号writeAccess； 阻塞mutexW信号，打印出开始写的信息并执行写函数write()；
3. 睡眠sleep持续时间lastTime；
4. 打印出结束写的信息并释放资源，阻塞互斥信号mutexW；
5. 阻塞互斥信号writeAccess，读者数量write\_count减1，检查write\_count，如果为0，那么释放信号mutexR。判断后解除互斥信号writeAccess。
6. 退出循环。



⑥ 创建读者线程函数：

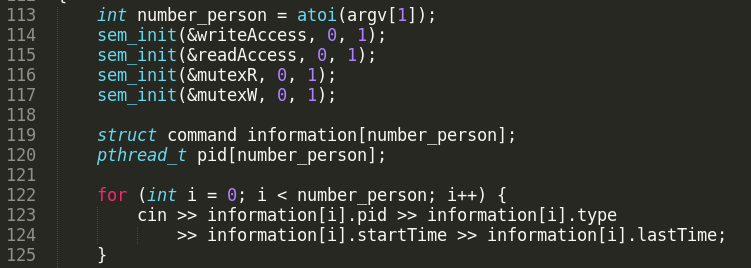
1. 将传入参数param转化为一个command的结构体，包含读者的pid（线程号），type（即是W，线程角色：读者），startTime（开始时间），lastTime（持续时间）。
2. 进入while循环：
3. 睡眠sleep开始时间startTime；
4. 打印出线程号为pid的线程正在申请资源信息，并开始阻塞mutexR检查读者是否允许读取，同时也阻塞信号readAccess；
5. 读者数量read\_count加1，检查read\_count，如果为1，那么阻塞mutexW。判断后释放互斥信号readAccess和mutexR； 打印出开始读取的信息并执行读函数read()；
6. 睡眠sleep持续时间lastTime；
7. 打印出结束写的信息；
8. 阻塞互斥信号readAccess，读者数量read\_count减1，检查read\_count，如果为0，那么释放信号mutexW。判断后解除互斥信号readAccess。

1. 退出循环。

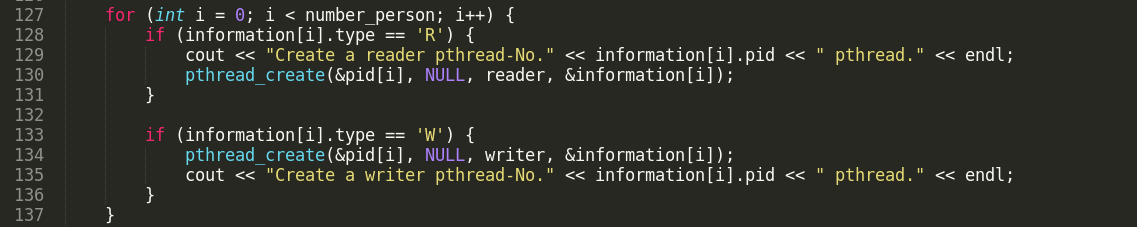


⑦ 进入main函数：

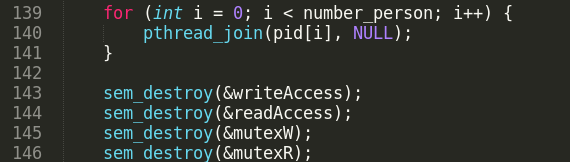
1. 创建信号量并完成初始化
2. 创建线程结构体
3. 完成指令的输入



⑧ 顺序遍历指令集完成线程的创建：

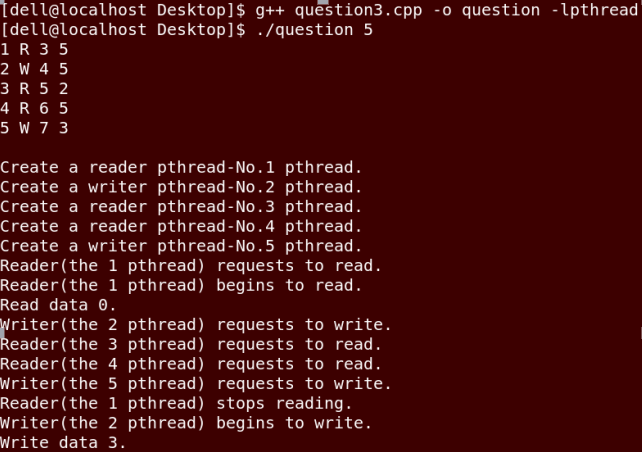


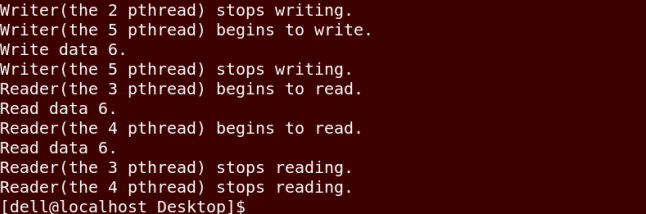
⑨ 非阻塞调用线程随后销毁信号量结束程序：



（3）程序测试与结果分析

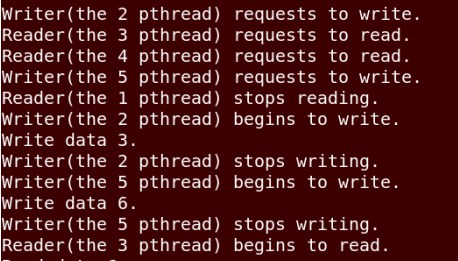
程序测试：





结果分析：

读者优先就意味着当有读者写者同时请求进行线程操作时候，先者要先进行操作，所以程序的运行结果中



这一段：写者线程2、读者线程3、读者线程4、写者线程5同时请 求操作，读者线程1由于是程序开始进行了读操作，所以率先终止，紧 接着写者线程2进行写操作，结束写操作，写者进程5进行写操作，结 束写操作，当这两个写线程操作完成之后，发出了stop指令之后，读者 线程3才开始进行毒操作，这也就体现了写者优先的操作，也就说明了 程序的正确性。

1. 实验心得：
2. 实验之前要理清楚老师给的资料中的内容，不要盲目的下手先做，否则只会是浪费了时间还没有什么收获。
3. 关于线程之间的互斥关系，通过这一次实验之后，也有了一些比较清楚的理解，同时也是对共享数据区的操作有了更熟练的掌握。
4. 同时实验中也处理了多线程之间的互斥和同步问题，再次遇到这类问题我们可以通过使用线程同步和互斥信号量来解决。
5. 这次实验也是自己第一次使用互斥信号量，相信在以后的学习中，会越发熟练掌握互斥信号量的使用。