**多线程编程**

**组名：The three stooges**

**小组成员：曾毓宁、徐锦畅、唐培津**

**一、实验目的**

1、掌握基于pthread线程库的多线程编程技术。

2、掌握基本的线程间同步技术(sem，mutex)。

3、理解共享资源并掌握其操作方法。

**二、实验内容**

**读者-写者问题多线程实现**

1、一个数据集（如数据、文件等）被N个线程读写；

2、一些线程只要求读数据集内容，称为读者 (Reader)，实验读者数不少于6个；

3、另些线程要求修改数据集内容，称为写者 (Writer)，实验写者数不少于3个；

4、多个读者可以同时读数据集内容，不需要互斥操作；

5、一个写者不能和其他写者或读者同时访问数据集，换句话说，写者和其他写者或读者之间必须互斥操作！

6、读者优先：如果有读者，写者需要等待！用于更新不频繁或更新影响不显著、读者较多的场合；

7、写者优先：如果有写者，读者需要等待！用于更新频繁或更新影响显著的场合。

**三、实验过程与结果**

**（一）读者优先**

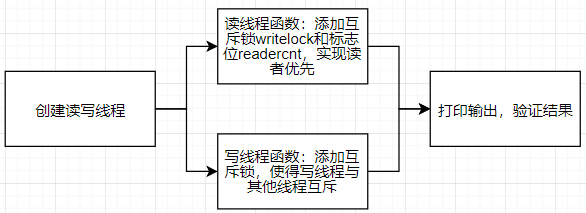
**1、读者优先的情况分析：**

（1）多个读者可以同时读数据集内容，不需要互斥操作；

（2）一个写者不能和其他写者或读者同时访问数据集；

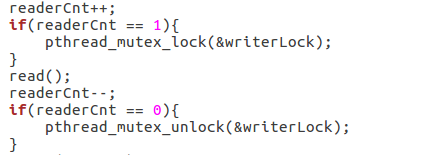
（3）如果有读者，写者需要等待。

**2、程序执行流程**



**3、程序实现**

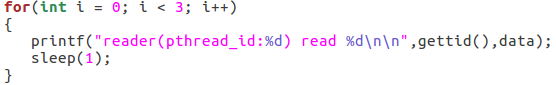
①在读者线程中，为了实现多个读者可以同时读数据集内容并且写者需要等待，在读者线程中用变量readerCnt计算此时访问数据集的读者数量，当第一个读者访问数据集，也就是readerCnt由0变1时给写操作加上互斥锁；当读者都退出后，也就是readerCnt变为0时对写操作进行解锁。



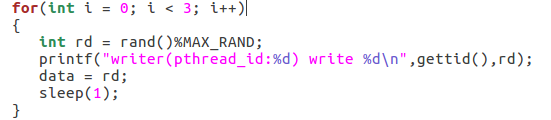
②要实现写者和其他读者写者均互斥，在写者线程中对写操作加上互斥锁。



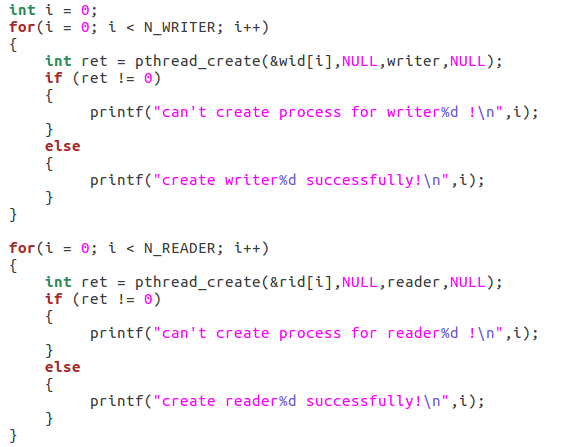
③一个读者线程在读取数据的时候需要进行三次读取。



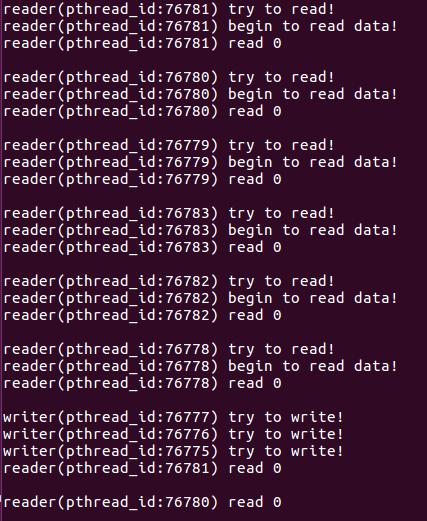
④一个写者线程在写数据时也是进行三次写操作，每次写入一个随机数数据。



⑤在主程序中创建3个写线程和6个读线程。



**4、运行结果**





观察运行结果，可以看到6个读线程同时读数据内容0，并且在读线程没结束之前写线程想要访问数据内容（比如writer(pthread\_id:76777) try to write!），但因为互斥锁存在，保证了在读线程没结束前写线程不得访问数据内容。在6个读线程都退出后，写线程才开始写数据。

**（二）写者优先**

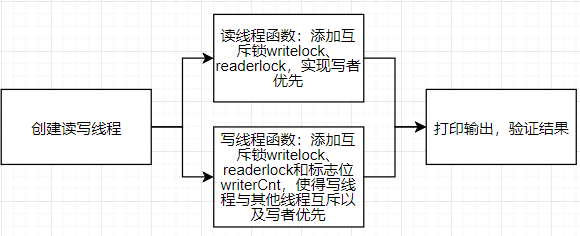
**1、写者优先的情况分析：**

（1）多个读者可以同时读数据集内容，不需要互斥操作；

（2）一个写者不能和其他写者或读者同时访问数据集；

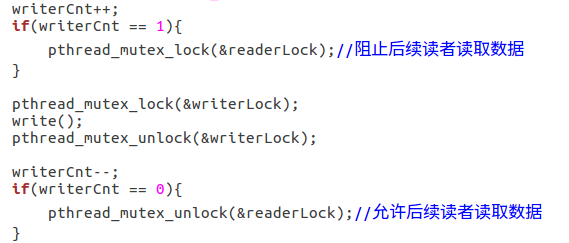
（3）如果有写者，读者需要等待。

**2、程序执行流程**



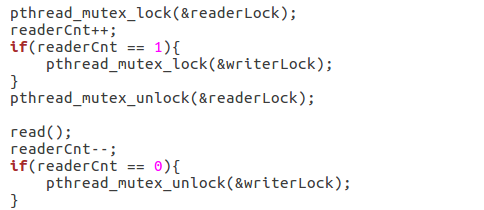
**3、程序实现**

①在写者线程中，为了实现一个写者不能和其他写者或读者同时访问数据集，并且在有写者时读者需要等待，这里用writerCnt计算此时访问数据集的写者数量，当第一个写者访问数据集，也就是writerCnt由0变1时给读操作加上互斥锁，阻止后续读者读取数据；同时在写者线程中对写操作加上互斥锁，使得当前这个写者不能和其他写者或读者同时访问数据集；当写者都退出后，也就是writerCnt变为0时对读操作进行解锁，允许后续读者读取数据。

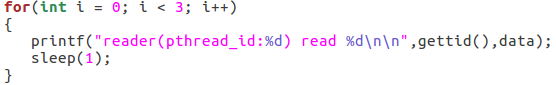


②在读者线程中，因为是写者优先，所以需要考虑当同时有写者和读者线程在等待时，在一个线程结束后，写者进程应当先进行，读者线程继续等待直到没有写者进程。

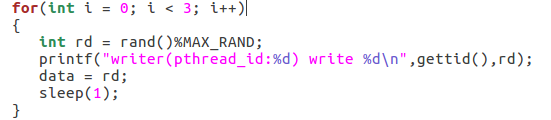
首先在读者线程中添加一把互斥锁readerLock，阻止其他正在等待的读者进入；当有读者正在读数据时给写操作上锁，阻止写操作修改数据；然后给读操作进行解锁，以免读操作不能同时执行；在读操作完成后，判断读者时是否都已经退出，也就是readerCnt是否为0，若读者都已经退出，则给写操作解锁，允许写者线程访问数据。



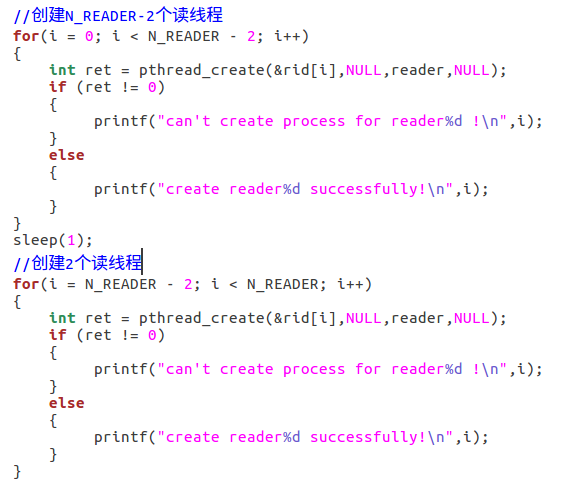
③一个读者线程在读取数据的时候需要进行三次读取。

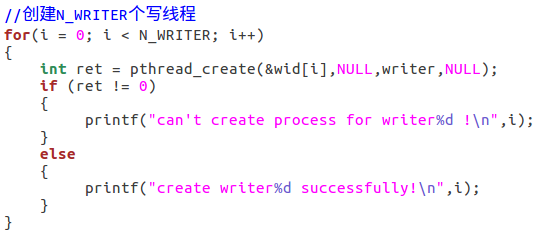


④一个写者线程在写数据时也是进行三次写操作，每次写入一个随机数数据。

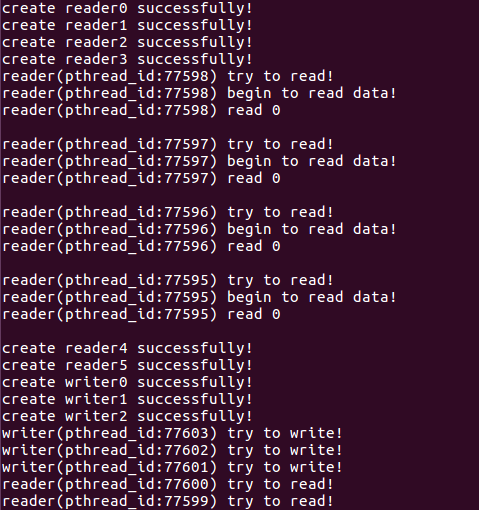


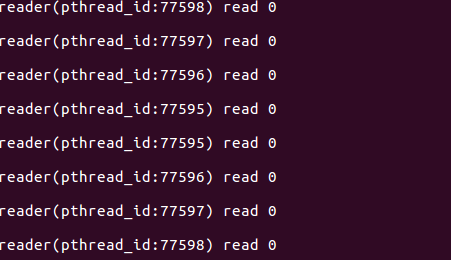
⑤ 在主程序中先创建4个读线程，判断多个读者可以同时读数据集内容；然后创建2个读线程和3个写线程，判断是读线程优先还是写线程优先。

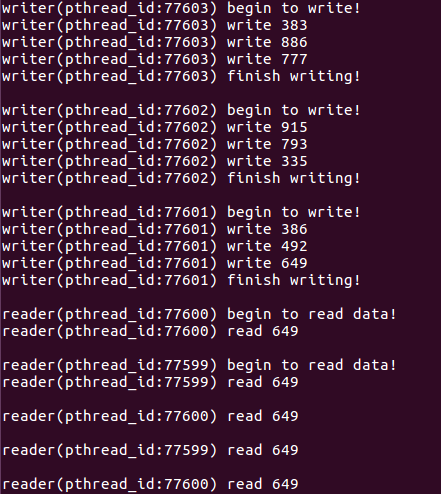




**4、运行结果**







观察运行结果，可以看到先创建4个读线程可以同时读数据集内容，在等待的2个读线程和3个写线程中，写线程优先，在写线程完成后读线程才运行。

1. **实验总结**

本次实验中，我们熟悉了多线程编程，学习了基本的线程间同步技术，为以后进行复杂的编程工程奠定了良好的基础。

1. **附实验源码**

**reader\_first.c:**

**/\***

**\* 多线程,读者优先**

**\*/**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <pthread.h>**

**#include <sys/syscall.h>**

**//#include <unistd.h>**

**#define N\_WRITER 3 //写者数目**

**#define N\_READER 6 //读者数目**

**#define W\_SLEEP 1 //控制写频率**

**#define R\_SLEEP 1 //控制读频率**

**#define gettid() syscall(\_\_NR\_gettid)//获取线程ID**

**pthread\_t wid[N\_WRITER],rid[N\_READER];**

**pthread\_mutex\_t writerLock = PTHREAD\_MUTEX\_INITIALIZER;//同一时间只能一个人写文件,互斥**

**int data = 0;**

**int readerCnt = 0;//正在进行读操作的读者数目**

**const int MAX\_RAND = 1000;//产生的最大随机数**

**void write()**

**{**

**printf("writer(pthread\_id:%d) begin to write!\n",gettid());**

**for(int i = 0; i < 3; i++)**

**{**

**int rd = rand()%MAX\_RAND;**

**printf("writer(pthread\_id:%d) write %d\n",gettid(),rd);**

**data = rd;**

**sleep(1);**

**}**

**printf("writer(pthread\_id:%d) finish writing!\n\n",gettid());**

**}**

**void read()**

**{**

**printf("reader(pthread\_id:%d) begin to read data!\n",gettid());**

**for(int i = 0; i < 3; i++)**

**{**

**printf("reader(pthread\_id:%d) read %d\n\n",gettid(),data);**

**sleep(1);**

**}**

**}**

**void \* writer(void \* in)**

**{**

**printf("writer(pthread\_id:%d) try to write!\n",gettid());**

**pthread\_mutex\_lock(&writerLock);**

**write();**

**pthread\_mutex\_unlock(&writerLock);**

**sleep(W\_SLEEP);**

**pthread\_exit((void \*) 0);**

**}**

**void \* reader (void \* in)**

**{**

**printf("reader(pthread\_id:%d) try to read!\n",gettid());**

**readerCnt++;**

**if(readerCnt == 1){**

**pthread\_mutex\_lock(&writerLock);**

**}**

**read();**

**readerCnt--;**

**if(readerCnt == 0){**

**pthread\_mutex\_unlock(&writerLock);**

**}**

**sleep(R\_SLEEP);**

**pthread\_exit((void \*) 0);**

**}**

**int main()**

**{**

**int i = 0;**

**for(i = 0; i < N\_WRITER; i++)**

**{**

**int ret = pthread\_create(&wid[i],NULL,writer,NULL);**

**if (ret != 0)**

**{**

**printf("can't create process for writer%d !\n",i);**

**}**

**else**

**{**

**printf("create writer%d successfully!\n",i);**

**}**

**}**

**for(i = 0; i < N\_READER; i++)**

**{**

**int ret = pthread\_create(&rid[i],NULL,reader,NULL);**

**if (ret != 0)**

**{**

**printf("can't create process for reader%d !\n",i);**

**}**

**else**

**{**

**printf("create reader%d successfully!\n",i);**

**}**

**}**

**sleep(20);**

**return 0;**

**}**

**//////////////////////////////////////////////////////////////////////////**

**writer\_first.c:**

**/\***

**\* 多线程,写者优先**

**\*/**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <pthread.h>**

**#include <sys/syscall.h>**

**//#include <unistd.h>**

**#define N\_WRITER 3 //写者数目**

**#define N\_READER 6 //读者数目**

**#define W\_SLEEP 1 //控制写频率**

**#define R\_SLEEP 1 //控制读频率**

**#define gettid() syscall(\_\_NR\_gettid)//获取线程ID**

**pthread\_t wid[N\_WRITER],rid[N\_READER];**

**pthread\_mutex\_t readerLock = PTHREAD\_MUTEX\_INITIALIZER;**

**pthread\_mutex\_t writerLock = PTHREAD\_MUTEX\_INITIALIZER;**

**int data = 0;**

**int readerCnt = 0, writerCnt = 0;**

**const int MAX\_RAND = 1000;//产生的最大随机数**

**void write()**

**{**

**printf("writer(pthread\_id:%d) begin to write!\n",gettid());**

**for(int i = 0; i < 3; i++)**

**{**

**int rd = rand()%MAX\_RAND;**

**printf("writer(pthread\_id:%d) write %d\n",gettid(),rd);**

**data = rd;**

**sleep(1);**

**}**

**printf("writer(pthread\_id:%d) finish writing!\n\n",gettid());**

**}**

**void read()**

**{**

**printf("reader(pthread\_id:%d) begin to read data!\n",gettid());**

**for(int i = 0; i < 3; i++)**

**{**

**printf("reader(pthread\_id:%d) read %d\n\n",gettid(),data);**

**sleep(1);**

**}**

**}**

**void \* writer(void \* in)**

**{**

**printf("writer(pthread\_id:%d) try to write!\n",gettid());**

**writerCnt++;**

**if(writerCnt == 1){**

**pthread\_mutex\_lock(&readerLock);//阻止后续读者读取数据**

**}**

**pthread\_mutex\_lock(&writerLock);**

**write();**

**pthread\_mutex\_unlock(&writerLock);**

**writerCnt--;**

**if(writerCnt == 0){**

**pthread\_mutex\_unlock(&readerLock);//允许后续读者读取数据**

**}**

**sleep(W\_SLEEP);**

**pthread\_exit((void \*) 0);**

**}**

**void \* reader (void \* in)**

**{**

**printf("reader(pthread\_id:%d) try to read!\n",gettid());**

**pthread\_mutex\_lock(&readerLock);**

**readerCnt++;**

**if(readerCnt == 1){**

**pthread\_mutex\_lock(&writerLock);**

**}**

**pthread\_mutex\_unlock(&readerLock);**

**read();**

**readerCnt--;**

**if(readerCnt == 0){**

**pthread\_mutex\_unlock(&writerLock);**

**}**

**sleep(R\_SLEEP);**

**pthread\_exit((void \*) 0);**

**}**

**int main()**

**{**

**int i = 0;**

**//创建N\_READER-2个读线程**

**for(i = 0; i < N\_READER - 2; i++)**

**{**

**int ret = pthread\_create(&rid[i],NULL,reader,NULL);**

**if (ret != 0)**

**{**

**printf("can't create process for reader%d !\n",i);**

**}**

**else**

**{**

**printf("create reader%d successfully!\n",i);**

**}**

**}**

**sleep(1);**

**//创建2个读线程**

**for(i = N\_READER - 2; i < N\_READER; i++)**

**{**

**int ret = pthread\_create(&rid[i],NULL,reader,NULL);**

**if (ret != 0)**

**{**

**printf("can't create process for reader%d !\n",i);**

**}**

**else**

**{**

**printf("create reader%d successfully!\n",i);**

**}**

**}**

**//创建N\_WRITER个写线程**

**for(i = 0; i < N\_WRITER; i++)**

**{**

**int ret = pthread\_create(&wid[i],NULL,writer,NULL);**

**if (ret != 0)**

**{**

**printf("can't create process for writer%d !\n",i);**

**}**

**else**

**{**

**printf("create writer%d successfully!\n",i);**

**}**

**}**

**sleep(20);**

**return 0;**

**}**