一、实验题目：

编写Linux下的多线程程序。熟悉多线程编程以及如何使用锁实现互斥访问。

二、实验目的：

1）熟悉pthread多线程编程：pthread_mutex, pthread_create, pthread_join, pthread_mutex，pthread_barrier等。

2）分析多线程编程中数据访问冲突问题

三、实验要求：

下载代码ph.c，在该代码基础上，修改或增加代码完成下列要求。

1、要求

1）解决数据丢失问题

2）使用barrier，解决ph.c代码中第92行的忙等待。

更详细的题目要求及说明，请参考后面的英文描述。

2、编写实验报告

依据实验报告模板，编写实验报告。实验报告内容包括：

1）设计思路说明；实现过程若依赖系统某些特性，请增加说明性描述文字。

2）请给出编译代码的具体命令，简单说明如何编译、运行、测试你提交的代码。如果程序由多个源程序构成，建议编写Makefile，或者给出编译脚本。

3）请给出带运行测试结果的截图

4）实验报告中要分析造成数据丢失的原因，为什么修改代码后就能解决问题。

3、作业提交要求

请将所有程序文件、实验报告、编译脚本（如果有）整体打包到单个zip文件，提交zip文件。zip文件名称：学号+姓名.zip，学号与姓名间不用添加任何符号，且学号在前。

四、补充材料

In this assignment you will explore parallel programming with threads and locks using a hash table. You should do this homework on a computer that has multiple cores. Most recent laptops have multicore processors.

Download ph.c and compile it on your laptop

注意：ph.c在附件中

```
$ gcc -g -O2 ph.c -pthread
$ ./a.out 2
```

The 2 specifies the number of threads that execute put and get operations on the the hash table.
After running for a little while, the program will produce output similar to this:

```
1: put time = 0.003338
0: put time = 0.003389
0: get time = 7.684335
0: 17480 keys missing
1: get time = 7.684335
1: 17480 keys missing
completion time = 7.687856
```

Each thread runs in two phases. In the first phase each thread puts NKEYS/nthread keys into the hash table. In the second phase, each thread gets NKEYS from the hash table. The print statements tell you how long each phase took for each thread. The completion time at the bottom tells you the total runtime for the application. In the output above, the completion time for the application is about 7.7 seconds.

Each thread computed for about 7.7 seconds (~0.0 for put + ~7.7 for get).

To see if using two threads improved performance, let's compare against a single thread:

```
$ ./a.out 1
0: put time = 0.004073
0: get time = 6.929189
0: 0 keys missing
completion time = 6.933433
```

The completion time for the 1 thread case (~7.0s) is slightly less than for the 2 thread case (~7.7s), but the two-thread case did twice as much total work during the get phase. Thus the two-thread case achieved nearly 2x parallel speedup for the get phase on two cores, which is very good.

When you run this application, you may see no parallelism if you are running on a machine with only one core or if the machine is busy running other applications.

Two points: 1) The completion time is about the same as for 2 threads, but this run did twice as many gets as with 2 threads; we are achieving good parallelism. 2) The output for 2 threads says that many keys are missing. In your runs, there may be more or fewer keys missing. If you run with 1 thread, there will never be any keys missing.

Why are there missing keys with 2 or more threads, but not with 1 thread? Identify a sequence of events that can lead to keys missing for 2 threads.

To avoid this sequence of events, insert lock and unlock statements in put and get so that the number of keys missing is always 0. The relevant pthread calls are (for more see the manual pages, man pthread):

```
pthread_mutex_t lock;     // declare a lock
pthread_mutex_init(&lock, NULL);   // initialize the lock
pthread_mutex_lock(&lock);  // acquire lock
pthread_mutex_unlock(&lock);  // release lock
```

Test your code first with 1 thread, then test it with 2 threads. Is it correct (i.e. have you eliminated missing keys?)? Is the two-threaded version faster than the single-threaded version?

Modify your code so that get operations run in parallel while maintaining correctness. (Hint: are the locks in get necessary for correctness in this application?)

Modify your code so that some put operations run in parallel while maintaining correctness. (Hint: would a lock per bucket work?)