Assignment 3:Threads and Locking

In this assignment you will explore parallel programming with threads and locks using a hash table. You should do this homework on a real computer (not xv6, not qemu) that has multiple cores. Most recent laptops have multicore processors. Submit your solutions on Google classroom.

Download ph.c and compile it on your laptop:

```
$ 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?)

Submit: your modified ph.c alongwith a README file that answers all questions above.