In-class: Threads and Locking

In this assignment we will explore parallel programming with threads and locks using a hash table. It is ideal to do this assignment on a computer that has a processor with multiple cores. Most recent laptops have multicore processors.

Please feel free to collaborate with others on these exercises.

Download ph.c and compile it on your laptop or Athena machine:

```
$ 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 output will be something along the lines:

```
0: put time = 2.871728
1: put time = 2.957073
1: lookup time = 12.731078
1: 1 keys missing
0: lookup time = 12.731874
0: 1 keys missing
completion time = 15.689165
```

Each thread runs in two phases. In the first phase it puts NKEYS/nthread keys into the hash table. In the second phase, it 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. You see that that completion time is about 16 seconds. Each thread computed for about 16 seconds (~3 for put + ~13 for get). This indicates that we achieved perfect parallelism; the threads didn't interfere with each other.

When you run this application, you may see no parallelism if you are running on a machine with 1 core or if the machine is loaded with other applications.

Independent of whether you see speedup, you will likely observe that the code is incorrect. The application inserted 1 key in phase 1 that phase 2 couldn't find. Run the application with 4 threads:

```
2: put time = 1.516581
1: put time = 1.529754
0: put time = 1.816878
3: put time = 2.113230
2: lookup time = 15.635937
2: 21 keys missing
3: lookup time = 15.694796
3: 21 keys missing
1: lookup time = 15.714341
1: 21 keys missing
0: lookup time = 15.746386
0: 21 keys missing
completion time = 17.866878
```

Two points: 1) The completion time is about the same as for 2 threads, but this run did twice as much work as with 2 threads; we are achieving good parallelism. 2) More keys are missing. In your runs, there may be more or fewer keys missing. There may be even 0 keys missing in some runs. 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 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