cores. Start by downloading the attached file, parallel hashtable.c to your local machine and you will compile it with the following command:

```
$ gcc -pthread parallel_hashtable.c -o parallel_hashtable
```

Now run it with one thread:

```
$ ./parallel_hashtable 1
[main] Inserted 100000 keys in 0.006545 seconds
[thread 0] 0 keys lost!
[main] Retrieved 100000/100000 keys in 4.028568 seconds
```

So with one thread the program is correct. But now try it with more than one thread:

```
$ ./parallel_hashtable 8
[main] Inserted 100000 keys in 0.002476 seconds
[thread 7] 4304 keys lost!
[thread 6] 4464 keys lost!
[thread 2] 4273 keys lost!
[thread 1] 3864 keys lost!
[thread 4] 4085 keys lost!
[thread 5] 4391 keys lost!
[thread 3] 4554 keys lost!
[thread 0] 4431 keys lost!
[main] Retrieved 65634/100000 keys in 0.792488 seconds
```

Play around with the number of threads. You should see that, in general, the program gets faster as you add more threads up until a certain point. However, sometimes items that get added to the hash table get lost.

Part 1

Find out under what circumstances entries can get lost. Update parallel hashtable.c so that insert and retrieve do not lose items when run from multiple threads. Verify that you can now run multiple threads without losing any keys. Compare the speedup of multiple threads to the version that uses no mutex – you should see that there is some overhead to adding a mutex.

You will probably need:

```
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
```

Once you have a solution to this problem save it to a file called parallel mutex.c.

Hint: You can also use man to get more documentation on any of these

Part 2

Make a copy of parallel mutex.c and call it parallel spin.c. Replace all of the mutex APIs with the spinlock APIs in pthreads. The spinlock APIs in pthreads are:

```
pthread_spinlock_t spinlock;
pthread_spin_init(&spinlock, 0);
pthread_spin_lock(&spinlock);
pthread_spin_unlock(&spinlock);
```

Do you see a change in the timing? Did you expect that? Write down the timing differences and your thoughts in a comment in your source file.

Part 3

For Part 3 continue working with parallel mutex.c. Does retrieving an item from the hash table require a lock? Update the code so that multiple retrieve operations can run in parallel.

Part 4

For Part 4 continue working with parallel mutex.c. Update the code so that some insert operations can run in parallel.