Homework #4 (due December 2, 2021, 11:59 PM)

Academic Honesty

Aside from the narrow exception for collaboration on homework, all work submitted in
this course must be your own.

Cheating and plagiarism will not be tolerated. If you have any questions about
a specific case, please ask the TAs. We will be checking for this!

Homework Notes

General Notes

- Read the assignment carefully, including what files to include. Don't assume limitations
  unless they are explicitly stated.
- Don't assume limitations unless they are explicitly stated.
- Treat provided examples as just that, not exhaustive list of cases that should work. When in
  doubt regarding what needs to be done, ask. Another option is test it in the real UNIX
  operating system. Does it behave the same way?
- Test your solutions, make sure they work. It's obvious when you didn't test the code.
- Make sure your assignment has been written and submitted on Anubis's IDE.
- Make sure to also submit a text document stating who your partner is, if you have one, to
  Brightspace. If you're working alone, submit a text document stating you did not have a
  partner for this assignment.

Parallel Hashtable with pthreads

In this assignment, you will take a non thread-safe version of a hash table and modify it so
that it correctly supports running with multiple threads. This does not involve xv6; xv6
doesn't currently support multiple threads of execution, and while it is possible to do parallel
programming with processes, it's tricky to arrange access to some shared resources. Instead
you will work on this assignment through Anubis's IDE which uses a multicore machine.

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 also 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 running 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 commands.
Alternatively, you can look them up the command online (https://linux.die.net/man/)

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 or submit them in a text document on Brightspace.

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.

Submission

Upload the 2 files you created onto Anubis.

1. Parallel mutex.c that you modified for steps 1, 3 and 4.

2. Parallel spin.c that you created for step 2.

3. Upload a text document onto Brightspace LMS detailing the NetID and name of your partner. If you're working alone, submit a document stating that you worked on this assignment alone.