CS/SE 4348 S19: Operating Systems

Exercise 3: Threads and Locking

Due date 13th March, 2019

Read the description below carefully. This is **individual** exercise. You cannot share your work with anyone. This exercise can be done in **cs1/cs2/csjaws.**

Parallel Programming

In this exercise, 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 processors/cores. cs1/cs2/csjaws have multiple processors/cores (verify by executing 'cat /proc/cpuinfo').

Copy the code tl.c from /cs4348-xv6/src/E3/ to your directory and compile it.

```
{cslinux1:~} gcc -g -O2 tl.c -pthread
{cslinux1:~} ./a.out 2
```

The argument 2 specifies the number of threads that execute put and get operations on the hash table.

After running for a little while, the program will produce output similar to this:

```
0: put time = 0.011398
1: put time = 0.011441
0: get time = 0.460219
0: 482 keys missing
1: get time = 0.464893
1: 476 keys missing
completion time = 0.476704
```

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/nthread 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 0.476 seconds. Each thread computed for about 0.47 seconds (~0.01 for put + ~0.46 for get).

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

```
{cslinux1:~} ./a.out 1
0: put time = 0.012343
0: get time = 0.852030
0: 0 keys missing
completion time = 0.864743
```

The completion time for the single thread case (~0.86s) is slightly less than twice the two thread case. Thus, the two-thread case achieved nearly 2x parallel speedup for the get phase on two processor/cores, which is very good. Why there is no speedup for put phase?

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 for 2 threads is roughly half that of single thread; we are achieving good parallelism. 2) The output for two 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 routines 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?) What do you observe?

A curious mind will repeat the above experiments for more than 2 threads (say, 10 or more.)
Write a brief report on what you inferred.

Make sure you comment the part of the code you added/modified.

Submission

Copy your modified tl.c source file and the report to the directory '/cs4348-xv6/xxxyyyyyy/E3/'.