Homework #4

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, its tricky to arrange access to some shared resource.

Instead you will do this assignment on a multicore machine.

Essentially any desktop machine made in the past 7 years or so should have multiple

1

Homework #4 (due May 2, 11:55 p.m)

NYU Tandon School of Engineering April 18 , 2021

Computer Science

CS-GY 6233, Spring 2021

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

2

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.

Submission

Upload the 2 files you created to NYU Classes:

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

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