Mason Haines
Cs 446
Threads part 3

PART III

1. Run looped_sum and threaded_sum a few times with whatever data file you like.

a. What output is the same when a program with the same data is run many times? Explain.

- The sum output or compiled data is the same regardless of how many times it is ran. All concrete constants are gonna remain the same regardless of there computer environment.

b. What output changes when a program is run with the same data many times? Explain.

- The total time taken for results or operations in code will vary. Different computers have different operating systems, CPUs, background operations etc which may cause variation in timing.

2. Run looped_sum with the tenValues.txt file many times.

a. Do you get the same sum when you run threaded_sum with ten_values.txt and no lock?

- Yes, when running both programs regardless of the lock the result is the same.

b. Do you get the same sum when you run threaded_sum with ten_values.txt and a lock?

- Yes.

c. How does the run time of looped_sum and threaded_sum (locked AND not locked) compare?

- The looped sum program has a significantly faster runtime than both the locked and unlocked threaded sum program. The faster of the threaded sum variations is locked.

d. Why is the total time to calculate the sum for the three cases different? Were they what you expected? Why or why not?

- The total time for each program varies based on user-selected thread count and locking mechanisms in threaded sum programs. Looping relies on system resources, while threading adds overhead. As expected, simpler looping performs faster due to reduced complexity in thread management.

Mason Haines
Cs 446
Threads part 3

3. Run looped_sum with the oneThousandValues.txt file many times.

e. Do you get the same sum when threaded_sum runs with oneThousandValues.txt and no lock?

- Yes.

f. Do you get the same sum when threaded_sum runs with oneThousandValues.txt and a lock?

- Yes.

g. How does the run time (in ms) of looped_sum and threaded_sum compare?

- The looped sum program has a significantly faster runtime than both the locked and unlocked threaded sum program.

h. Why is the total time to calculate the sum for the two programs different? Were they what you expected? Why or why not?

- Similar to the ten.txt case, total time varies in each program based on user-selected thread count and locking mechanisms in threaded sum programs. Looping utilizes system resources, while threading introduces overhead. As anticipated, simpler looping performs faster due to reduced thread management complexity. If there was a significant difference in the smaller data set, I would expect it to be magnified for the larger data set.

4. Does the use of a lock in a threaded program have any impact on performance? How does the number of threads and the amount of data affect the performance of the threaded program with and without locks?

- The use of locks does have a performance impact. The time taken is is much faster than if I had been unlocked. From what I noticed the lower the thread count the faster the unlocked became. At one thread it was even faster than the locked by at least 50%. With the different datasets I did not realize much of a difference in performance speed.

5. Is the lock necessary to compute consistent values every time the program is run? Why or why not? Why do you think that occurs? You should run the program with

and without a lock and with a few different data files to get the full picture.

- From running the program multiple times without locks I would say that the lock is necessary to compute consistent values. Randomly the value would be wrong for unlocked and I have not experienced that for the threaded lock. This is most likely because the mismanagement of resources and poor syncing of timing.

6. What considerations decided what was the Critical Section? Explain.

Where the threads had access to the most information and were modifying it. Essentially the bottle neck of the code which ended up being in the sumArray function.