Name: Alexander Young
Section: 7
University ID: 169253104

Project 2 Report

Summary:

10pts

In this project, we created a simulated “bank” where we could get and update the amount of currency that an account had. The program was multithreaded, with the worker threads actually doing the account activities. A goal was to have the main program never block on reading requests. The amount of worker threads and accounts was variable, and provided by the runner. We were to have each of the accounts protected by a mutex (or the whole bank for part 2), as access and updating an account was not thread safe. A provided test script simulated a user of the program, and ran rapid file commands against our program.

Part II:

6.2:

5pts Average processing times for TRANS and CHECK requests (from test script):

For Fine Grained:
Total wait time for the 400 TRANS requests: 322.432 (s), average 0.806 (s) per request
Total wait time for the 1000 CHECK requests: 10.935 (s), average 0.011 (s) per request

For Coarse Grained
Total wait time for the 400 TRANS requests: 4117.899 (s), average 10.295 (s) per request
Total wait time for the 1000 CHECK requests: 4212.100 (s), average 4.212 (s) per request

6.3:

3.2.1:

3pts Which technique was faster - coarse or fine grained locking?

Fine grain locking was faster. However, my coarse grained implementation always produced exactly the right final account details, the total balance, and the ISF’s matched exactly with what was expected.

3pts Why was this technique faster?

Fine grain is faster as each account can be locked individually, which allows multiple threads to simultaneously work, if they don't need the same account. Coarse grain is definitely simpler to implement though.

3pts Are there any instances where the other technique would be faster?

I think that when doing a lot of operations that involve a large amount of what is being protected by the mutex, then locking the whole thing would be more efficient, as then there is only one mutex to lock, instead of several. For instance, truncating a database table. It is more efficient to lock the whole table, than lock each row.

3pts What would happen to the performance if a lock was used for every 10 accounts? Why?

This is interesting. The performance could benefit, if multiple sequential items are frequently used together. For instance, if rows 1 through 10 are almost always used at the same time, this could be a performance benefit. However, if the data is not sequential, this can degrade performance, as then unused items are unnecessarily locked.

3pts Discuss the probable "optimal" locking granularity (fine, coarse, or medium)?

As with everything, it depends. If the data being locked is sequential, then medium or coarse can be a benefit. If all of the data is needed, then coarse is definitely the way to go. However, if the data is random and nonsequential, then fine grained is the best bet.