Name: Jonathon Schnell
Section: 2
University ID: 077633581

Project 2 Report

Summary:

10pts

I learned quite a bit while writing the code for this project. There were several functions that needed to be written to implement a queue of requests with a mutex. This was a great review for me on the implementation details of a queue. I also got practice mutex locking for thread safe operations such as dequeing requests and mutex locking for each account. This took some time to perfect and avoid deadlocks.
I also learned about the differences between fine and coarse grained mutex locking. Coarse grained was easier to implement because there is less chances for a deadlock. The disadvantage is that it is slower than a fine grained solution. This matches what I expected going into this project.

Part II:

6.2:

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

Fine:
Trans: 528.783 seconds (1.322/request)
check: 0.027 seconds (0.00/request)
coarse:
Trans: 3895.373 seconds (9.738/request)
Check: 0.024 seconds (0.00/request)

6.3:

3.2.1:

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

Fine grained.

3pts Why was this technique faster?

Fine grained is faster because it allows more parallelization, many requests can be processed at one time so long as all the requests are for unique accounts. This is not possible in the coarse grained program, in this program a single mutex must be acquired to process requests meaning that parallelization is not allowed, children must wait on a single mutex.

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

Coarse grained will, in a best case scenario, only be able to match the speed of a fine grained program. I cannot think of a situation other than a deadlock that will make coarse grained faster than fine grained. I guess one way that it may be faster is that we don't need to order the requests in ascending order, this is not required for coarse grained because this will never cause a deadlock.

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

We would probably see a result somewhere in between fine and coarse. This is because some requests can be processed in parallel but not all. This is because we may be locking accounts that we do not need for the transaction in question. By doing this we could be causing other children to wait for a mutex that they could otherwise acquired independently to process in a fine grained solution.

Optimal may be a good solution for certain programs but in this case the best scenario is to have a mutex for each account so each request can potentially be processed in parallel. An optimal algorithm does have the advantage of being slightly less complex than fine grained, there are less mutexes to store in memory and keep track of. There may be other advantages to breaking up mutexes into chunks of resources depending on your application, such as it being easier to avoid a deadlock.

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