Name: Jonathan Hess

Section: 1

University ID:

**Programming Project 2 Report**

**Summary (10pts):**

I feel like I learned a lot from this project. The most important skill was debugging threaded programs. The progress on this project was halted by a segmentation fault I could not debug via print statements or file outputs. Using Visual Studio Code’s C++ debugger I was able to place a breakpoint and look at the structure of my queue. The issue was that the next pointer was not being assigned NULL in the initialization. I would recommend a lab or reference for a similar debugging tool breakdown.

The code works by creating a queue of requests. The requests are created in the main function via the request.c functions. This queue is then passed to worker threads that execute the operations on the accounts. The locking mechanism I originally created locked all accounts when performing multiple reads allowing to guarantee correct transactions.

**6.2:**

**(5pts) Average runtime for each program (use the “real” time)**

The fine with a few too many ISF transactions had timing of:

Initial deposits (100 TRANS) took 3.1 seconds to finish, script waited 20 seconds.

Random transactions (300 TRANS) took 3.3 seconds to finish, script waited 30 seconds.

-- Request Wait Time --

Total wait time for the 400 TRANS requests: 548.351 seconds, average 1.371 seconds per request

Total wait time for the 1000 CHECK requests: 13.304 seconds, average 0.013 seconds per request

The mixed with accurate sum and transactions had a timing of:

-- Script Run Time --

Initial deposits (100 TRANS) took 10.3 seconds to finish, script waited 20 seconds.

Random transactions (300 TRANS) took 10.7 seconds to finish, script waited 30 seconds.

-- Request Wait Time --

Total wait time for the 400 TRANS requests: 2011.860 seconds, average 5.030 seconds per request

Total wait time for the 1000 CHECK requests: 14.489 seconds, average 0.014 seconds per request

The coarse program had a timing of

-- Script Run Time --

Initial deposits (100 TRANS) took 30.3 seconds to finish, script waited 20 seconds.. Might need a higher [wait\_time\_initial]

Random transactions (300 TRANS) took 40.1 seconds to finish, script waited 30 seconds.. Might need a higher [wait\_time\_final]

-- Request Wait Time --

Total wait time for the 400 TRANS requests: 8724.508 seconds, average 21.811 seconds per request

Total wait time for the 1000 CHECK requests: 7354.899 seconds, average 7.355 seconds per request

**6.3:**

1. **(3 pts) Which technique was faster – coarse or fine-grained locking?**

The fine grained was much faster than the coarse locking.

1. **(3 pts) Why was this technique faster?**

The majority of the time executing (waiting) is done in the bank “read\_account” and “write\_account”.

1. **(3 pts) Are there any instances where the other technique would be faster?**

If locking the individual accounts took longer than the access time with sparse edits.

1. **(3 pts) What would happen to the performance if a lock was used for every 10 accounts? Why?**

The result would vary based on variables. The less threads or the more accounts this more this affect would speed up the locking and performance. But in our example we would expect a small increase in performance over the fine because we have 1000 accounts but only 10 threads. Assuming a uniform distribution the .

there shouldn’t be problems with addressing or locking the accounts. This means that the time because of having 1000 accounts shouldn’t be much different than 100 locks. If the locking time is nearly the same, then the speed can only decrease because of locking. As the account count becomes larger more advanced forms of locking will be needed.

1. **(3 pts) What is the optimal locking granularity (fine, coarse or medium)?**