Name: Joseph Schmidt
Section: B
University ID: 174945787

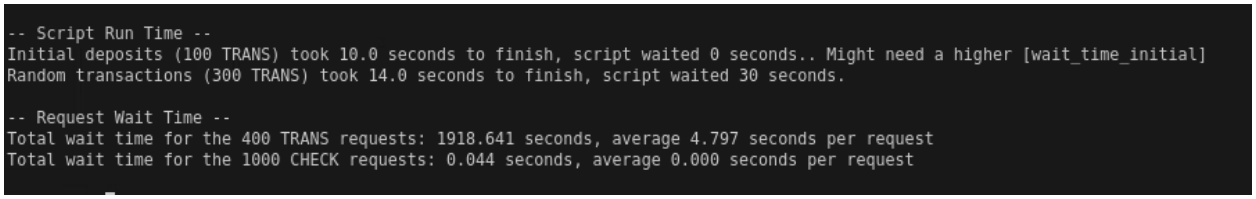
Programming Project 2 Report

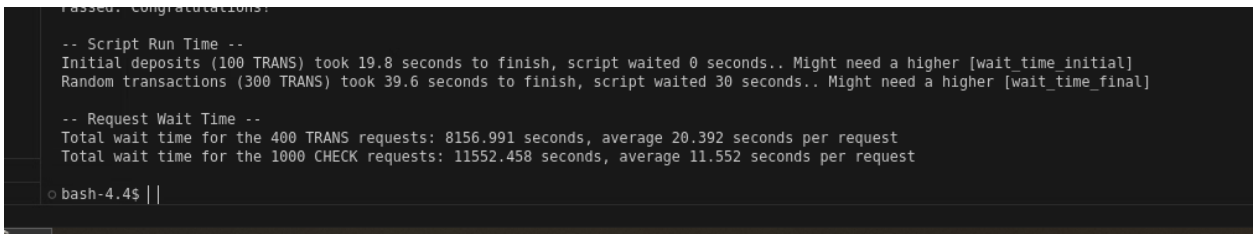
Summary (10pts):

In this lab we worked on creating a bank account that handled requests with multiple threads. Whenever a request was sent in it was put onto the tail end of a linked list. This linked list was protected by a mutex lock. Threads were in a continuous loop waiting for the queue to not be null then would lock the queue and take off the request. Right before giving up the queue lock the thread would grab an account lock. This means it would be the only thread accessing an array of locks. This is confusing at first but ensures that this thread won't have to wait for other threads when trying to obtain all of the account locks. A thread would have to obtain multiple account locks when there is a TRANSACTION request where multiple accounts can make a transaction at the same time. If this transaction is void, i.e. an account goes negative at the end of the transaction, the transaction would be tossed, and all values would be returned to what they were before the transaction took place. Along with transactions, the program had to take CHECK requests where a thread would check the current value of an account. Like mentioned before there is an array of mutex locks, one for each account. Only one thread would be able to grab locks at a time. This was the account lock from above. This ensures no out of order requests occur where threads hand off locks to each other making it so that transactions don't take place out of turn.

Overall, this lab was a great experience for learning more about the C programming language, about operating systems, and about multithreaded programs.

6.2:
(5pts) Average runtime for each program (use the "real" time)





6.3:

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

Fine-grained locking.

2. **(3 pts) Why was this technique faster?**

This was because in the coarse-grained locking scheme there was a lot of time spent waiting for the one lock to be unlocked.

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

Were there is a minimal amount of threads that there aren't many threads waiting for the one lock and it avoids the overhead of having a thread for each account.

4. **(3 pts) What would happen to the performance if a lock was used for every 10 accounts? Why?**

It would probably be a middle ground between the two. It has the problem of coarse-grained locking where there will be 10 accounts waiting for one lock. It won't be as severe as the complete coarse-grained though as there are only 10 locks waiting.

5. **(3 pts) What is the optimal locking granularity (fine, coarse or medium)?**

Medium. This is more practical instead of a dedicated lock for each account. This could lead to a massive number of mutex locks being implemented when the number of accounts gets to a high number. This could lead to memory or other issues. It is better to cut this down by a factor of 10 or more for a very small hit in performance.