Name: Vaibhav Malhotra

Section: F

University ID: 582047495

**Project 2 Report**

**Summary:**

**10pts**

In this project, we got a lot of experience with threads and programs. I learned a lot about how threads work together in getting the code done more efficiently. I also learned more about structs and arrays. I had to figure out how to make objects so I looked more into structs and how to implement them. I also learned more about how to take input strings and convert them into arrays so you can dissect them. In the end, I think I learned quite a bit more about C and about threads. It was a very challenging project for me, one which I spent a lot of hours on but the project ran successfully.

**Part II:**

**6.2:**

**5pts** Average for each program:

appserver: 1m16.108s, 1m16.269s, 1m16.068s -> 1m16.148s

appserver-coarse: 1m18.074s, 1m17.684s, 1m18.057s -> 1m17.938s

**6.3:**

**3.2.1:**

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

Fine-grained technique was faster

**3pts** Why was this technique faster?

It has much more concurrency and isn’t as wide as coarse. Instead of locking more than you need in coarse, you only lock what you need making it faster.

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

If you have to lock so many things, then there is more overhead in setting up fine grained. So, coarse might be a better option.

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

It might be faster than a lock for every account, but you could have issues with overwriting and deadlocks.

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

I think that for a case where the number of accounts is not a lot larger than the number of

workers than coarse would work the best. If it was an extremely large bank with few

worker threads than I think fine would work better. I think medium would not really work

for either case unless the transactions were very sequential in accounts for example

TRANS 1 100 2 100 3 100 4 100