Name: Nihaal Zaheer
Section: 2
University ID: 105254206

Programming Project 2 Report

Summary (10pts):

The project is about writing a multithreaded program that simulates the management of a set of bank accounts. When the server is started, a fixed number of bank accounts are created, and users can perform transactions on these accounts as well as query an account's current balance.

The program must use multiple threads to service the requests simultaneously to keep response times small. User requests are made via the command line, and the server program records the request and presents the user with a transaction ID. The requests are processed in the background, and the results of the requests are printed to a file to avoid interfering with user input.

The program should contain two types of threads, a "main" thread, and one or more "worker" threads. The main thread initializes the server state and creates all other worker threads. Each worker thread services one request at a time. After servicing a request, the worker thread will service another request, and so on until the server exits. The number of worker threads is specified as an argument on the command line.

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

It takes around 50 seconds to a minute

6.3:

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

In general the use of coasre or fine grained locking depends on the users needs. Coarse grained locking can be simpler to implement and manage, as there are fewer locks to manage, which can lead to better performance if the application spends a lot of time acquiring and releasing lock, but threads may need to wait for the lock to be released even if they only need to access a small portion of the data protected by the lock.

Fine-grained locking, on the other hand, can lead to higher concurrency, as threads can access different parts of the data without waiting for a lock on the entire data structure, but it can be more complex to implement and manage, as there are more locks to manage and the potential for deadlocks and other synchronization issues.

But for this project, fine grained locking was faster.

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

For coarse grained locking, it locked the entire bank during balance check and transaction which is not very efficient. But with fine grained locking you pick and choose what to lock in each scenario which is much more efficient. That is only locks the relevant accounts at a given time.

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

If we had a smaller code where it would not make a difference if a whole section was locked, then coarse grained locking would be a good idea to use, since it also would be easier to implement.
For example, in this project, if we wanted to perform functions on all the bank accounts at the same time, then coarse grained locking would be a good idea to implement.

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

There would be a delay or to put it better, the performance would be affected by this. If we lock an account that need not be locked, and then lock it again, it would take longer to execute that it usually would.

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

   It depends on usage, I would say a mix of both, or medium grain locking would be better. This is so that we can choose what to use where and how much effeciency we are willing to give up.