CMSC 621 Document for Project 1:

This is implementation Centralized Multi-User Concurrent Bank system. The entire source code is available in 2 files, bank.cpp and client.cpp. Also Makefile is included in the same folder with two targets, compile and clear to create object files and remove object files respectively. It generates two object files server and client.

After generating server object file following is the process to execute :

```
./server [port] [Relative path of file which has initial values of customers]
```

After generating client object file following is the process to execute :

```
./client [server Address] [Service port] [Time step] [Path of Transaction.txt]
```

Program Working:

When server is executed it creates a data structure for customers from reading Records.txt file. An object of customer class is created for each customer. Properties of this object are account number ,customer name, balance. Vectors are utilized to create objects dynamically as customers connect to server.

A mutex is also initialized when object is created for that customer. Therefore there are n mutex for n customers. Two functions called lock and unlock are defined in class utilized to do the same. These mutex eliminate race condition problems and ensures correctness of data.

Then socket is setup and listens for connection requests in the port given as command line argument. When a client connects to server, thread is created and function serve is called. In this function server keeps waiting for customer to send its transaction requests. After receiving request it will process it and reflect the changes on datastructure. When transaction is successful acknowledgement is sent to client or reason for failure is mentioned.

At client side requests are sent according to time stamps. Request rate can be easily varied using multiplier time stamp variable which is passed as command line argument for client.It produces a time delay equal to multiplier times of difference of current time stamp and previous transaction time stamp. Therefore request rate can be easily varied using this variable.

Time taken for each transaction is measured by using gettimeofday function. This function is called before transmitting transaction request to server and called again after receiving acknowledgement from server. Difference between these two values is measured and transaction time is calculated in micro seconds.

Transaction time for each transaction is stored in array and later used to calculate average transaction time. After all the transactions are completed total transaction time and average transaction time is printed. These values are used to plot graphs and analyzing performance of program.

Possible Improvements:

Atomicity of the transaction is not ensured. That is when program is made to crash(by killing the process) during a transaction involving transfer of funds from one account to another system might go in to unstable state due to partial transaction. There is no mechanism to undo invalid transactions.

Also there is no mechanism to make changes to data structure permanent. Therefore changes made are lost when program is closed.

It is also possible to implement security measures like encrypted password for each customer and security questions.