**Project 1**

**Advanced Operating Systems**

**Ajinkya Wakhale**

In this programming assignment, we were asked to implement a Centralized multi user concurrent bank Account manager which should be multithreaded with two components: Client and Server

For my design, I have created two C files, **client.c** and **server.c**.

Server.c use set of records at the server initially which is used as a input file i.e Records.txt that server reads. Similarly, client issues request to server from the file Transactions.txt which has all the transactions line by line which server will process.

**Design:**

Server creates a socket, binds its address to it and then wait for the incoming connections. Similarly, client creates a socket and then tries to connect to server.

To support multithreading and concurrent execution, for each client which successfully connects to server, a thread is created which has its handler where all the business logic resides. In my program, it is **create\_thread**. So, each client, a thread will be generated. This supports multi-threading.

Similarly, once a client is connected to server, it reads transactions line by line from Transactions.txt and sends it to server line by line. There is a wait time for each transaction, for that I have put timer which will wait for that much time and then will send the transaction to server.

At server side, once the client is connected server will load Records.txt into array of structures. Structure array has all the records.

Client will first send number of transactions to server and will receive acknowledgment that it has received the number of instructions. For each line of transaction received from client, it will check the records line by line to check if the account number in transaction matches with the record and if it matches, it will do the required transaction checking the balance. If the transaction is of **withdraw**, first it will check whether the account has required balance and if not it will send back the message to the client saying **not sufficient balance.** Also, if the transaction has account number which is not in the records, it will send the message saying **no account found**. And if all goes smoothly, it will send back the client **old balance** and **new balance**.

For concurrent execution, locking mechanism is used to make sure that two process does not work on the same data at the same time. So, before every withdraw and deposit, a lock is implemented and once the data is updated, the lock is released. Locking mechanism makes sure that at run time, the transaction gets correct value.

Once the client executes all its transactions, server will close the connection to client.

To compile the program-

Client- **gcc -o client client.c**

Server- **gcc -o server server.c -pthread**

To run the program, first server is started with required port number

Server- **./server 7867**- 7867 is port number

Client-**./client localhost 7867 0.1**- localhost is address of server. I designed both on local system and hence it is localhost and 7867 is port on which server is running and 0.1 is wait time for each transaction.

To compile and run the program from **Make file-**

**1. Make**

**2. Open new terminal-**

**command: make client\_start.**

**Issues and prospects:** As it is reading each records line by line, performance issues come. Also, it is difficult to synchronize at server side for the timestamp. Improvements could be done in designing and coding to improve the time taken by the program to execute. In future, for each transaction it could write on Records.txt and then the program will become complex from reading the file and synchronizing for each transaction. Also, as it is a Bank account manager, security issues should also be handled. Also, if number of transactions are more it would be advisable to divide the work load into multiple threads and then synchronizing.

**Client Output file**- test1.txt for one client

**Server output file** – server\_output.txt