Project 1: Design Document
By: Sofia Panuganti (sofiap1@umbc.edu)

Project Requirement:

Bank Server: The server program that services online requests for account manipulations and maintains all customer records correctly.

Clients: Customers are clients of the bank server and use its services to update bank accounts. The operations that can be performed on an account are: withdrawal of an amount from an account and deposit of an amount into an account. Additionally, the bank server can have its own service that periodically deposits an interest amount to each account based on some fixed rate.

Overall Program Design:

This project is implemented using C++ on Ubuntu 20.04.3

Server-side program: The server reads from a text file which contains account number, name, balance amount (space-separated) in each line to create account information and stores it in vectors. A socket is created with appropriate socket system call. The socket structure is initialized. The host address (server address) is bound to the socket created and the server starts listening for up to a hundred connection requests from clients. Upon accepting a connection request from the client, the server sends an acknowledgement that connection has been successfully established. A separate thread is created for each client using POSIX Thread model, and a thread handler function *operations_handler()* is assigned to each client to perform the transaction requests: **Withdraw**, **Deposit** and **Balance Enquiry.**

The server receives a file from the client side which contains the transactions requests information in the format: timestamp, account number, transaction type (withdrawal/deposit/balance enquiry), amount (space separated). The server compares account number information to ensure the account number received from the client exists in the records database. The transaction request does not occur if this comparison returns false. If the account exists, based on the type of transaction specified with the letter **'w'** for withdraw, **'d'** for deposit, **'b'** for balance enquiry, the server calls the appropriate function to carry out the transaction. For a withdraw (w) transaction, the account balance is subtracted with the amount

mentioned; if the amount exceeds balance in the account, a message 'insufficient balance' is sent to the client. For a deposit (d) transaction, amount is added to the balance and an acknowledgement is sent to the client. For a balance enquiry (b), the balance of the customer's account is displayed on the server side and a 'statement generated' acknowledgement is sent to the client. The section of the code in the server-side program where the records database is being accessed by the client for transactions is enclosed in a mutex lock to avoid race conditions - simultaneous access to the same record. After there are no more transaction requests left, the client socket connection is closed.

The server program is killed with an interrupt signal from the keyboard (**SIGINT, CRTL+C**) and the records database is updated once the server is killed.

Client-side program: The client is executed with command-line arguments: hostname, port number to connect to the host/server. The client reads transaction details from a Transactions.txt file to send to the server line-by-line. A client-side socket is created, and a request is sent to the host to establish connection. Upon server accepting the request, connection is established between the server and client. The transaction details sent to the server are processed according to the timestamps held by each query, in order to avoid race conditions. If the mutex lock is acquired by a process for executing a transaction, the next process will sleep until the executing process releases the mutex lock. A clock is incorporated to calculate the time taken for each transaction to execute to evaluate performance.

Contents of the submission:

src Directory:

1. server.cpp
2. client.cpp
3. Design Documentation
4. Output Screenshot
5. makefile

Results Directory:

1. Performance Results Document

Compiling and executing the code using makefile:

A makefile is present in the src directory with the following commands for compiling and executing the code. Note: The command line arguments (hostname, port no) have default values in makefile for 'make clientRun' command which can be changed by editing in the makefile.

Compile server and client programs: **make compile**

Run server program: **make serverRun**

Run client program: **make clientRun**

Clean all objects created by make file: **make clean**

Design tradeoffs:

Once the client connection has closed and no more data is being received or sent, the server must be terminated manually by sending a signal interrupt from the keyboard (CTRL+C). Upon termination of server, all the changes to the records database are correctly updated and reflected at once.

Additionally, timestamps are manually sent for each transaction. The banking system does not handle transactions with the same timestamps, timestamps in a descending order.

Possible improvements and extensions:

As a project extension, a simple banking system GUI can be created.

Message queue can be incorporated to service multiple clients, multiple servers with a finite number of threads in the server dequeuing the message queue and executing the requests.

Clock synchronization could be used to omit manually sending timestamps for each transaction by maintaining the at which each request is made. For handling transactions with the same timestamp, poll() function could be used.