Multithreaded Bank System:

I designed a bank system in the *programming language C* which kept track of and altered people's accounts as needed.

Each account is a struct which contains a char* (the name of the account), a float (the current balance of the amount), and an int (signals whether or not that particular account is currently being accessed. 0 if it is not. 1 if it is.)

I implemented this system using a socket

***The port number this program uses is "5389"

Initially when you start running the **server (server.c). Two threads are running the "main thread" and the "accepting connections thread" (which is a kernel thread created by the main thread)

The server's main thread prints out a complete list of all accounts every 20 seconds. (I accomplished this by using sleep(20) in the server's main thread).

The information printed for each account will include the name on the account, the account's current balance, and the words "IN SERVICE" if a client is accessing that account.

Anytime a client connects to the server the "accepting connections thread" creates a new kernel thread which handles the interaction between that client and the server.

***When the **client.c** file is run. It must be given 1 command line argument, which is the name of the machine where the client expects to find the server. If for some reason the client is unable to connect to the server you provided (like the server is not currently running for example). It will keep trying to connect to that server every 3 seconds.

VALID COMMANDS THE CLIENT CAN USE:

- Open `accountname`
- Start `accountname`
- Credit `amount`
- Debit `amount`
- Balance
- Finish
- exit

The **open** command opens a new account for the bank. It is an error if the bank already has a full list of accounts, or if an account with the specified name already exists. A client in a customer session cannot open new accounts, but another client who is not in a customer session can open new accounts. The name specified uniquely identifies the account. An account name will be almost 100 characters. The initial balance of a newly opened ac count is zero.

The **start** command starts a customer session for a specific account. **The credit, debit, balance, and finish commands are only valid in a customer session.** It is not possible to start more than one customer session in any single client window, although there can be concurrent customer sessions for different accounts in different client windows. **There cannot be concurrent customer sessions for the same account.** It is possible to have any number of sequential client sessions.

The **credit and debit** commands add and subtract amounts (respectively) from an account balance. Amounts are specified as floating-point numbers. Either command complains if the client is not in a customer session. There are no constraints on the size of a credit, but a debit is invalid if the requested amount exceeds the current balance for the account. Invalid debit attempts leave the current balance unchanged.

The **balance** command simply returns the current account balance.

The **finish** command ends the customer session. Once the customer session is terminated, it is possible to open new accounts or start a new customer session.

The **exit** command disconnects the client from the server and ends the client process.

Other Useful Info:

I used 3 mutex locks for this system.

I used the first one to prevent clients from opening new Accounts while the server was printing out information. And also to prevent clients from opening two accounts with the same name. aka *if 2 clients want to open an account with the same name at the same time.*

The second I used to make sure that two clients could not access the same account at the same time.

The third I used to prevent two distinct client from getting the same id from the server. i.e to make sure that the server would communicate with each one properly.