Roberto Merino

Mina Gadallah

Multithreaded Bank System

This project is divided in two pieces.

1-server

2-client

In our Server:

We have one thread that responds back to the client when the client sends a request to the server. Our Server holds all the data and bank accounts.

The threads were created asynchronizaly which means that threads don't interfere with one another. There is no need to wait for an input on the client, therefore the server can send a message, and otherwise, both sides will have to wait.

When the client sends a command to the server, we Use our Tokenizer to split the command String into tokens and we analyze the token based on the possible valid commands on the server. Such that {Open, Start, debit, credit, balance, Exit} followed by the account name.

We check if the string construction is valid. If it's valid we proceed, otherwise, we prompt an error "invalid command"

Based on the command we proceed, when we run the server, it runs with 0 accounts. Client can use open accountName to open a new account and that creates an account on the server with everything initialized to 0.

If the client chooses to start, they must use Start and then accountName. And then they are able to use the other commands that are related to start {debit, credit, balance}

Validations

Client will not be able to open an account while another client is opening an account as it's going to prompt you to try again.

Client will not be able to access an account while the same account is open by another client

Client will not be able to open a non-existed account as the server will prompt a message "account doen't exist"

Client cannot use an existed account name to open a new account. It will not allow him!

The server max is 20 accounts and you can't over load more than two accounts.

If the server disconnect, the client gets disconnected automatically.

In our Client:

Our client is the user interface, which the user uses as a tool to connect to the server. It’s as simple as a terminal that takes commands and sends them over to the server.

We have two threads in our client. The First one sends the command to the server as inputs.

The second thread takes care of the messages coming from the server to the user as a result of the command that is being initially sent by the first thread.