Asst 3 Documentation

Anna Godin
Tiffany Moral

Design

- Client Side
    - Error checks the commands, makes sure the commands are properly spelled. If incorrect, there is a while loop that checks for validity.
    - When client types quit, the command is sent to server to disconnect, as well as a flag is set on the client side to close the sockfd.
    - If the client cannot connect to the server immediately, it tries 4 more times, and then times out after that.
    - Once a connection is established, the client side spawns a thread to handle the reading from the server
    - The original thread handles writing to the server
- Server Side
    - Initializes mutexes and semaphores
    - Creates socket, binds and then starts an infinite while loop to listen for connections. For every connection made, the server spawns a thread with the unique sockID to do all the bank operations
    - Threads are detached
    - There are two buffers, one to read and one to write.
    - Each command has its own function
    - Creating accounts is mutex protected, since only one account can be created at a time.
    - When an account is served, it locks itself so no other account can serve it

Difficulties

- Originally figuring out how to connect the client to the server
- First we had it on localhost but then had to transfer it to work over a network address
- When we spawned threads, we had trouble with concurrency when multiple clients wrote to the server at once.
- We also had trouble with locking the database with semaphores, but figured it out in the end

Testing Procedure

- My partner and I tested the synchronization and concurrency abilities of the program by running the serve on one of our computers, and then a client on each, and then we sent commands to the server at the same time to see how it would handle it

Thread Synchronization Requirement

- <u>Threads</u>
    - Client - client->server writing thread
    - Client - server->client reading thread
    - Server - spawns a thread for each new connection

    - Server - thread for diagnostic output
- <u>Semaphores</u>
    - Used to lock the database when printing diagnostic output
    - Sem_wait when starting to print, and then sempost when finishing to print, to unlock the database so the threads can use it
    - Sem_wait when receiving the commands and sempost after command is finished
- <u>Mutexes</u>
    - Adding accounts is mutex protected. When an account is attempted to be added, there is a mutex lock in place so that only one can be added at a time
- <u>Synchronization</u>
    - All the above method contribute to the synchronization of the database and the whole server side of the program
- <u>Signal Handling</u>
    - When the client ctrl C, the server catches it and disconnects that sockfd, but keeps running
    - When the server ctrl C, then all clients receive the SIGINT signal and shut down gracefully