CS214 Programming Assignment 5

Vincent Xie, Jacob Rizer

Multithreaded Bank

Our implementation of the multithreaded bank has a server and a client. The server is run and uses port 6799, which was chosen arbitrarily. It listens until clients connect to it and prints out the bank info every 20 seconds using a timer. If an account is being currently serviced, it prints out "IN SESSION" next to the account. If there is nothing in the bank, then the bank info will be empty. Every time it gets a new connection, the accepter thread accepts it and creates a new thread. Each thread has a socket file descriptor to its own client, which each thread can read from and send data to.

The client takes in an argument, which is the hostname for the server. The client also connects using port 6799. It creates a socket and, connects the socket to the server. If the server is not found, then it tries again in three seconds until it connects. Once it connects then it creates two threads. The first thread is a thread that reads input from stdin (the terminal) and then sends it to the server. The client only allows one command to go through to the server every 2 seconds. **You can input more commands within that two-second time frame, however the first of the commands in the queue will not get sent until 2 seconds are up***.* The second thread is a thread that reads messages that are sent from the server and outputs them. When "finish" is entered, the client closes its session with the account and when "exit" is entered, the client closes its session with the account (if any) associated with the client, closes the connection with the server and exits.

Back on the server end, if an invalid command is sent, then the server sends a message back to the client that tells it that the command is invalid. If a valid command is sent, then it calls the respective command. First of which is open, when open is called, we first make sure that we get the mutex lock for the bank. This is to prevent opening a new account while the bank is being printed. The same happens for the other commands. Likewise, when calling start on an account, it makes sure that it has the mutex lock for the account. **(Extra Credit: If for any reason it does not have the lock, it tries again in 2 seconds.)** This prevents two accounts from accessing the same account at the same time. For the other functions, such as debit, debit, credit we make sure we have the lock for the account before making changes to the values or printing them out. To prevent deadlocks and race conditions, we made sure to lock the locks in the same canonical order.

(Extra Credit: We have also bundled a version of our bank that takes advantage of multiprocessing and shared memory. In the server, we would create a new process using fork() instead of creating a new thread. So, there is a main process, which deals with the connections to new clients, and each child process would have its own client, which it will read commands from and send data to. Also, since we are using multithreading, we created shared memory that the processes could share using a memory map. The memory map creates a new file, which holds the bank data. The processes can then share this memory and read and write to the bank data.)