Gavriel Tsioni (Section 01) and Daniel Tsioni (Section 02)
Systems Programming 198:214

PA5 (Multithreaded Bank System) Readme

Solution Design Description

Our design is as follows:

<u>Client.c:</u>

Client takes an IP address as an argument. The client will first try to establish a connection with the server with this IP over the port defined in network.h. It will retry every 3 seconds if it is unsuccessful. It then launches two threads. One thread will wait for responses from the server and display them to the user. If this response is ever “end” then the client will close. The other thread waits for input from the user and sends this input to the server after checking it lightly for validity. The client also has a signal handler in place for SIGINT which will send a ‘finish’ command to the server when the client closes. This is to ensure that if a client closes then it will end any active session it might have with the bank server.

By having two threads (one for responses and one for input) we ensure that the server has constant communication with the client. Server responses are shown to the user in real time. This also allows the server to shut the clients down at any time.

The input thread throttles user input by waiting two seconds in between each input. This makes sure that the user cannot send input more than once every two seconds.

<u>Server.c:</u>

The server first starts the client acceptor thread. This thread will wait for clients to establish a connection with the server and then it will spawn a client service thread. It also lets the client know if it connected to the server successfully, and prints any relevant error messages regarding connection issues.

The client service thread takes an FD as an argument. This FD is what allows us to communicate with a client specifically. It then monitors for any input from the client. If input is received from the client, it will pass it to the interact function. It then takes whatever response the interact function generates and sends it back to the client.

The interact function is what actually changes the bank. It checks the input for validity like correct number of args and correct operations. It also handles the mutex locking of accounts. It uses the functions stored in account.c to manipulate the bank.

Mutexes are handled in the interact function. Alongside the bank array is an array of mutexes with a one-to-one relationship. The start portion of interact locks mutexes when a session is started (or tells the user they cannot start a session with the given account yet because the mutex is already locked). The finish and exit portions of interact unlock those mutexes.

The FD's for the current clients are stored in a linked list. This allows the server to message all active clients at once.

A signal handler is in place for SIGINT. When the server receives a SIGINT signal, it will go through the linked list of all connected accounts and send them "end", which will close the client process.

By multithreading the server in this way, we can have multiple clients interacting with the server at one time. However this presents certain issues, such as two threads manipulating the same data. We solved this issue using mutexes, which helped us ensure thread synchronization. Accounts cannot be opened by multiple threads at the same time, using a mutex lock to ensure this. A single account cannot be started by multiple threads, which is also ensured by mutex locks. Also the balances cannot be printed while an account is being opened.

Account.c

Account.c holds functions which can manipulate a bank. There is an account struct which holds an accounts relevant data. A bank is an array of pointers to accounts, and is held in server.c. Account.c can debit, credit, balance, and open accounts.