Assignment 3

Multithreaded Bank System

Systems Programming Fall 2018
Professor Francisco
Created by: Alfonso Buono

1. Abstract:

The objective of this assignment was to test our knowledge and capabilities of threads, synchronization, signals, and sockets by having them all work together in a text-based online banking system.

2. Introduction:
    a. For this project there were two aspects that were worked on. There was both a client and server program in order to communicate and hold information. The client program would connect to the server and send over different commands and information in order to get the desired results. The responses to these commands from the server are then printed to STDOUT in order to let the client know what was done or any errors if there existed any. The server program has to handle multiple clients connecting to the server. It keeps track of different account and their balances. Furthermore, the server checks the commands sent by the client to see if they are correctly formed and formatted. Whether or not it is a proper command and the command was successful, it would send a response to the client, either confirming the command worked, or telling there was an error. Furthermore, the server also prints out all of the accounts and their information every 15 seconds to STDOUT. Also, when the server receives a Ctrl+C it handles the sigINT signal and disconnects all clients and closes all sockets and ends all threads.
3. Methodology:
    a. General Overview:
        i. Server:
            1. For this project I essentially made the server handle a majority of the aspects. For example, the server handles command checking, seeing if commands are formatted correctly or are valid. When initialized, the server creates a thread that's only job is listening for any attempts made by a client to connect to the server. When a connection is made, a new thread is spawned that handles that new client and the thread_id is stored in a global list (this is used later when I have to join all threads when a Ctrl-C is caught). A message is also printed to STDOUT indicating that a client has connected. Then, the newly spawned client thread handles the

commands sent to the server by the client. Once a command is received, it is checked to see which of the commands, all of which are listed below (along with their formatting and rules), it is and the specific action is performed if permitted. Depending on if the command occurs or not, the server then writes back to the client either saying their command was a success or if there was an error when using that command. Furthermore, the server also writes to STDOUT whenever a connected client disconnects The server also has a timer that is set when it is first created. This timer throws a SIGALRM every 15 seconds. During this time, account creation is disabled, and all current accounts, their balances, and whether or not they are in session are printed to STDOUT on the server program. Once it is done printing out all of the data, account creation is permitted again. The server takes in no commands, but when Ctrl-C is pressed, the server catches the SIGINT. Once it catches this signal, it disconnects all of the connected clients, joins all of the client threads that are running, and handles all of the threads, sockets, and data before terminating itself.

ii. Client:

1. For this project I made the client very light-weight and offloaded a lot of the work to the server. Therefore, the client only has 2 main parts besides the initial connecting to the server. Since either the client or server can be run first, the client repeatedly tries to connect to the server every 3 seconds if it does not connect. However, besides this the other parts are the two threads that handle input and output separately. This allows the client to receive information while sending information as well as allows the server to send the client information even if the client is stuck in its 2 second period where it cannot send commands. How the I/O was handled is discussed more in depth below.

b. Client I/O:

i. Input:

1. For my project I defined input as what the client sends to the server. This was done by when creating the connection to the server, a thread was spawned and constantly reads from STDIN. Since read is blocking, it waits until a user inputs a command/string and hits enter. Once the command is entered the thread then writes the command to the server. After this, it then sleeps for 2 seconds in order to prevent the client from sending multiple commands too quickly. After the 2 seconds, the thread then reads again and waits for client input

ii. Output:

        1. For my project I defined output as what the client receives from the server. This was done by when creating the connection to the server, a thread was spawned. This thread then constantly reads and waits for a message from the server to be passed. Once a message is received, depending on what the message is it either outputs it directly to STDOUT, or disconnects the client from the server and terminates the program.

c. Commands:
    i. Command handling was done on the server side. For example, the server checks whether or not the command is a valid command, and whether or not the following characters are applicable to the command. All commands must be typed in lower-case. Below is a list of the commands and how they are used:
    ii. Create:
        1. The syntax of the create command is as follows: create <accountname>
        2. Create must be in all lowercase followed by a space and then followed by <accountname>. <accountname> can be any visible ascii character, and the only limitations are that it cannot be an accountname that already exists on the server as well as it cannot exceed 255 characters long. Furthermore, create cannot be used when the client is currently serving an account. The user must "end" before being able to create. Also, create will execute but not right away when an account is being created by another client or when the server is printing out all of the accounts and their information. Instead, the command will execute after the other action is finished. If an account is successfully created, the server will send "Account created" to the client which will display the message.
    iii. Serve:
        1. The syntax of the serve command is as follows: serve <accountname>
        2. Serve must be in all lowercase followed by a space and then followed by <accountname>. <accountname> can be any visible ascii character, and the only limitations are that it cannot exceed 255 characters long, the accountname must be a valid accountname that already exists in the server, and the account associated with the name must not be in session by another client. Furthermore, serve cannot be used when a client is currently serving another account. The user must "end" before being able to serve another account. Depending on the error, the server will respond with an appropriate message to the client in order to properly use the command. It will also respond with a

confirmation if the account successfully becomes in service.

iv. Deposit:
    1. The syntax of the deposit command is as follows: deposit <amount>
    2. Deposit must be in all lowercase followed by a space and then followed by <amount>. <amount> can be any non-negative double. Furthermore, deposit can only be used when an account is in service, if not the server will send the client a message prompting the user to serve an account first. Upon a successful completion, the server will prompt the client that the money was successfully deposited.

v. Withdraw:
    1. The syntax of the withdraw command is as follows: withdraw <amount>
    2. Withdraw must be in all lowercase followed by a space and then followed by <amount>. <amount> can be any non-negative double that is equal to or less than the current balance of the account that is in service. If a user is attempting to withdraw more than they have, the server prompts the client that they have insufficient funds. Furthermore, withdraw can only be used when an account is in service, if not the server will send the client a message prompting the user to serve an account. Upon successful completion, the server will prompt the client that the money was successfully withdrawn.

vi. Query:
    1. The syntax of the query command is as follows: query
    2. Query must be in all lowercase followed by NO other characters. If there are any characters following "query" a message will be sent from the server to the client prompting the user to use query correctly. Furthermore, query can only be used when an account is in service, if not the server will send the client a message prompting the user to serve an account first. Upon a successful completion, the server will display the current balance of the account that is in service by the client.

vii. End:
    1. The syntax of the end command is as follows: end
    2. End must be in all lowercase followed by NO other characters. If there are any characters following "end" a message will be sent from the server to the client prompting the user to use end correctly. Furthermore, end can only be used when an account is in service, if not the server will send the client a message prompting the user that no account is in service. Upon a successful completion, the server will display a message

indicating that the current session/service has been ended and the account can now be serviced by any other client.

viii. Quit:
1. The syntax of the quit command is as follows: quit
2. Quit must be in all lowercase followed by NO other characters. If there are any characters following "quit" a message will be sent from the server to the client prompting the user to use quit correctly. Quit CAN be called even in serving an account. If serving an account, the server will switch the account to not in service, as well as send a message to the client indicating that the client is disconnecting. Then the client will disconnect. If the client is NOT in a service session, the server will just send a message indicating that the client is disconnecting.

d. SIGNAL HANDLER:
   i. A signal handler was created and used in order to catch both SIGINT and SIGALRM. When Ctrl-C is pressed on the server, the SIGINT signal is caught by the signal handler. From there, the server then sends a message to all clients and then disconnects all of them. After this, the server then joins all of the threads that were working with each client as well as the thread that was accepting new connections. It then clears all of the accounts and then after closing all sockets terminates the program.
   ii. The signal handler also was used to catch SIGALRM. This signal occurs every 15 seconds as a timer was set when the server program was first initialized. After the signal is caught, the server then locks all of the accounts and prevents any accounts from being created. It then prints out all of the account information as follows: Account name, followed by a tab, account balance, followed by a tab, and then IN SERVICE if the account is in service. Once it is done printing out all of the information it then unlocks all of the accounts.

4. Testcases:
   a. Testing the assignment was in small parts. At first I tested to make sure that the client and server could connect on the same computer. I then checked to make sure that the client and server could connect if they were run on different computers. After this, I tested each of the commands, their syntax, and the rules regarding when some of them can be run. After this was done, I tried connecting 10 clients and was able to do so successfully. Besides testing the commands, I believe creating testcases for this assignment is odd as it is just attempting to do different commands from different clients and see how the server handles it.
5. Submission Files:
   a. The files that I am submitting are as follows: Makefile, server.c, client.c, and bank.h.
   b. Makefile:

        i. The Makefile compiles both server.c and client.c and creates their respective executables.
    c. Server.c:
        i. This file contains all of the functions and aspects that are described for the server program. It utilizes thread creation, signals, as well as sockets in order to allow connections from multiple clients.
    d. Client.c:
        i. This file contains all of the functions and aspects that are described for the client program. It utilizes two threads in order to handle input and output separately as well as allows for input command throttling (can only send a command every 2 seconds).
    e. Bank.h:
        i. This file is the header file that is shared between both server.c and client.c. It holds all of the include statements, as both programs utilize very similar header files, structs and defines some variables that are used or were used for testing.

6. Running the Program:
    a. In order to make the two executables, bankingClient and bankingServer, just run the "make" command. Once these two executables are created this is how you run each program.
    b. bankingServer:
        i. ./bankingServer <PORT>
        ii. <PORT> is the port that you will be using. It should be consistent with the port that you are using when running the client in order for the two to connect.
    c. bankingClient:
        i. ./bankingClient <IP> <PORT>
        ii. <PORT> is the port that you will be using. It should be consistent with the port that you are using when running the server in order for the two to connect.
        iii. <IP> is the IP that the server is being hosted in. For example, if the server is being hosted on the "decorator" iLab, then <IP> should be decorator.cs.rutgers.edu.