Salman Omer, Seyeon Park ASST 3

The main server thread is initialized to open the first socket for communication on the input port. Then it enter an infinite loop to constantly check for new client connections. When a new client connection is found, it creates another socket for that connection for use with that particular client. After this socket is made, the program spawns a thread to service that particular client in the function singleClientHandler(). This function works in two phases. First it waits for a message from the client with recv() and then it sends back a response with send() after processing. Since recv() is a blocking call, the particular thread waits until the client either disconnects or sends a message. This two step process is encompassed by an infinite loop so that the particular client can send as many messages as it wants.

The server handles the logic for account creation, serving, ending, querying, depositing, and withdrawing. This logic occurs right after one of the server threads recieves a message from the client. The create method makes a new account in a master linked list of accounts. This method is mutex protected so that multiple threads do not make an account at the same time which would mess up the linked list data. The create, deposit, and withdraw methods are protected by a semaphore. This means that none of these methods can fully execute if the initial server thread is printing out the diagnostic information. This ensures that the diagnostic information is correct as of a certain time. After any one of these commands are processed by the server, the server creates a unique return message variable to return information to the client. This can either be an error if the process could not be completed or it could be a completion message. Then the server sends that message back to client. If the client disconnects from the server, the thread is closed and all resources are dealt with.

The server has to implemented signal handlers, one for sigint and one for sigalarm. The sigint handler occurs when ctrl-c is inputted in the console. This function closes all threads by using a global linked list of tids which is populated whenever a new thread is created. After each thread is closed, the function closes the main socket which is using the port and terminates the program. The sigalarm handler is triggered by the setitimer() function which triggers the alarm every 15 seconds. The sigalarm handler conducts a sem_wait() on the semaphore so that none of the threads can create, deposit, or withdraw. Then the handler prints out all the diagnostic information. Finally it execuets a sem_post() and returns.

The client uses two threads as per the project specifications. After establishing a connection with the server after multiple attempts using the input arguments, it creates one thread to send messages and one thread to recieve messages. The thread to send messages waits for a user input followed by a 'return' key press. Once this press is registered, the input is checked to be a valid input and then it is sent to the server using the send() command. After the command is registered, the thread invokes the sleep() command so that another input cannot be processed for 2 seconds. However, the user can still write in the console, this is unavoidable. But, if the user presses return during those two seconds then the command will not be processed until after the two seconds are other. The second thread is the recieve thread. This thread invokes the recv() function on a loop, waiting for a message from the server. The server is set up to only send a message back after the client sends something, so this recv() call in

the thread is guaranteed to complete execution after the send() call in the other thread. The client is also set up to recognize the sigint signal, causing it to free all memory and close all file descriptors. The threads of the client do not use any critical region since no data is stored in the client at any point