Our code starts with a server or client being created. If a client is created first with no server up, it throttles 3 seconds before attempting to connect to the server socket, continuing until it successfully connects. If the server starts first, we use listen() to make it a passive socket that runs accept() on an infinite loop in order to continually except new client sockets. This loop will terminate on SIGINT. Every time a new client socket connects, we add its unique sock file descriptor integer to a global linked list. If Control+C is ever entered on the server side, it will automatically get handled by are termination signal handler, where it will go through this global linked list of client sockfds and send quit 1 by 1 to each of them, which will then gracefully close the client socket and quit gracefully on the client side.

When the user give input, it first checks on the client side if the input is valid or not. If it isn't, it goes right back to waiting for user input without 2 second throttle. If it is a valid command, it will write the command to the server socket in the clients sender thread, and do the appropriate command on a global linked list we have stored on the server side. Server will check some additional error checks, such as account already in session, no account currently in session to end, etc. Upon successful or unsuccessful command execution, we write back to the clients receiver thread describing what happened. In the background, we used setitimer() to send a SIGALRM in real time every 15 seconds, which gets sent to our print_sig_handler. We then use a binary create a thread and use a binary semaphore to lock the global linked list so that while the list is printing, if another client inputs a command, the integrity of the list of accounts and info will not be compromised by conflicts. After printing, the semaphore will sempost(), and thus synchronize all data to global memory, where all client and server threads are able see the same linked list. Upon quit message on client side, or Ctrl+C on server side, all infinite loops break, and then join all threads that were created, thus maintaining synchronous relations.

Difficulties: properly throttling input command (as well as connection to server), printing accounts every 15 seconds, quitting gracefully on client side when CTRL+C is inputted on server side, implementing a semaphore that worked as intended, making sure that all our sockets closed upon completion.