Asst3 Documentation
Shyam Patel Phurushotham Shekan
CS 214
Prof. Francisco

Testing

1. Run make
2. the .out files are names bankingClient and bankingServer
3. run

Operation

1. The client essentially connects to the socket and creates 2 threads: 1 for reading and 1 for writing. This ensures that the client will always be listening and sending information to the server.
2. The first issue was obtaining the value from 2 part inputs. We ended up coming with the solution of using the strstr function from string.h and delimiting by a space. We would then memcpy() the the returned pointer+1 to the end of the char* into a temp variable.
3. Once we handled that issue, the rest was relatively simple since the setting up the sockets was straight forward. A simplified and high level perspective of what our code does is:
   - The server will start up and connect to the client at Port No. 11111
   - The client will provide commands which we will parse to see if they are valid: if invalid they are prompted to provide another input, if valid then the command goes through. The client cannot input commands more than once every 2 seconds.
   - We maintain a linked list of structs which store the necessary info for each account
   - Every 15 seconds a SIGALRM signal is sent and threads are locked in order to print out the information for all accounts in the global linked list.

Structure

- We used the following structures:
  1. account: stores the information for each account
  2. account list: kept track of the number of accounts as well as head and tail of the linked list of account structs
  3. connection: kept track of a thread's mutex, cond ref, thread ref as well as some flags and file descriptor
  4. connections: kept track of the link list of connection struct's head and tail references
  5. clientArgs: just used to pass void* args
  6. socketfd: simply used to pass the int arguments in the client side when the thread is created

Functions

- **I couldn't use underscores for the variable names because latex interprets that as a mathematical equation so I used a white space instead of an underscore**
- void handlesigalarm(int) : essentially iterates trhough all connections and sends a flag to all threads in order to pause them to print he info. Then prints info and resumes threads
- suspendMe(mutex, int) : changed the flag to 1
- resumeMe:sets the flag back to 0 and notifies all waiting threads to wake on provided wait condition
- void checkSuspend(mutex, pthread cond, int) : checks if thread is suspended, if not suspended then uses condition wait to wait thread
- void handle sigint(int) : closes all open sessions, joins threads and exits
- int startServer() : sets up sockets, starts the 15 second sigalarm cycle, connects to client
- void trimNewLine(char* buffer): gets rid of extra newline character void deleteConn(conn t) : destroys the given connection
- void addConnection(conn t) : adds connection
- conn t * findConnection(int): find the connection based on a file descriptor from the connt list
- conn t removeCOnnection(int) : removes connection from conn t linked list
- account t* findNode(char*) : finds given account struct that matches char* input
- account t* addToList(account t*) : adds a given node to the account struct linked list
- char* printData(account t*, char*): prints all info for a given node
- void printDiagnostics() : prints info for all accounts
- void* commandHandlet(void*): reads messages from the client and sends them to the server if valid
- account t* processClientRequest(char*, account t*): parses the client side command and acts on it as well as error checks it
- void* readServer(void*) and void* readServer(void*): **client side functions** that are called in the two threads created to ensure that the client is always reading and writing to the server