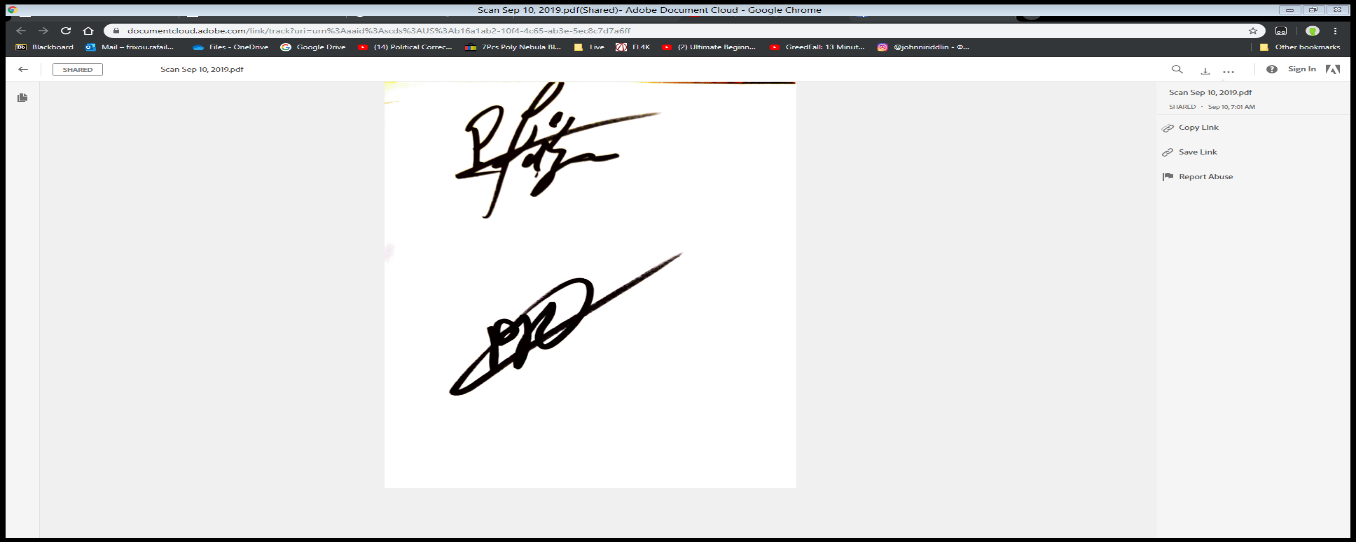
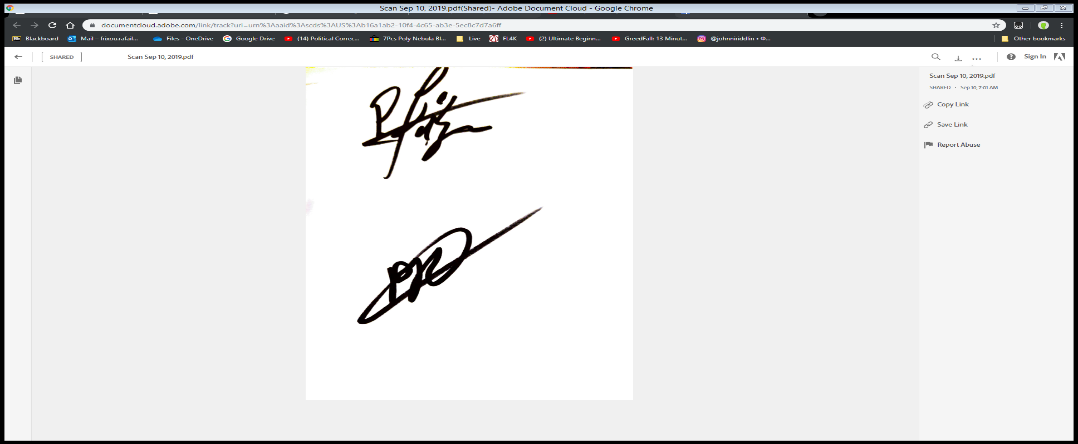
ECE 316 FINAL PROJECT

University of Cyprus [2020]



We hereby consent that this is our own work:

Rafail Frixou Panayiotis Procopi

# General Overview:

For the final project for this class we were asked to create a chat application using pthreads on Linux. The chat server is responsible for providing a mechanism for the clients to communicate. This includes allowing clients to login so they can be identified by username and forwarding messages from one client to another. This also includes listing the usernames of the clients that are currently logged in to facilitate communication. The chat application allows multiple clients to connect to a chat server.

Contrary to the guidelines provided (“The server must accept and maintain connections to all clients and relay chat messages between them.”) we have decided to implement a direct connection between the clients requesting to chat. The connection of both client <-> server and client <-> client was achieved using socket programming in a hybrid c/c++ implementation.

In detail the server is responsible of establishing a connection and responding to the following commands:

* login <username>: to establish a connection
* list: to provide all connected \*to the server\* clients
* initiate <name>: to establish a direct connection through a different port to the client <name>
* active: since we have opted for a single terminal implementation this command was considered unnecessary; thus we chose not to include it
* bye: to terminate both the client<->client and client<->server conversations

The client is responsible of acquiring the user’s input and relaying it to either the server or user according to the state of our implementation.

Structure:  
Server:

The server code consists of a driver Main function and a thread operating a connection handler function. The driver main is responsible for setting up the server socket and accept all connection requests on a predefined port. When it accepts the request it sets up a new socket and starts a new connection handler thread to accept and manage the client’s requests.

## Client:

The client code consists of a driver Main function and two threads each respectively responsible for a write handler and a read handler function. The driver main is responsible of setting up the connection to the server and upon successful connection to initiate the two aforementioned threads and their respective functions. The write handler is responsible for reading the user’s input and relaying it to the connected socket (either server or client) and the read handler is responsible of receiving that socket’s input and displaying it to the user.

## In detail:

### Client

Main:

* Create socket and run the connect command. The socket will try to establish a connection through the hardcoded port defined.
* If the connection is established:
  + Procude two threads as follows:
    - read\_handler
    - write\_handler
* Else print an error message and return 1;

Read\_handler:   
\*The read handler is responsible for reading the messages relayed by the server or the messages received from the initiated client and printing them to the user’s screen\*

* The thread created reads the socket buffer and transfers it to another string.
* If the command has the format: *"(init) <" + curr\_user + ">" + " |" + mes\_port + "|"* then the client creates a soocket and expects to find a server to connect to through the port (mes\_port) defined.
  + The current user is saved inside a gloaval string variable, and it is either the server or the user with whom a connection has been initiated.
* If the command is bye then the thread closes the socket connection and returns to the server if it is connected to another client or closes communication completely and retuns to main.
* If the command is none of the above then the read\_handler prints the command to the user’s screen.

Write\_handler:  
\*The write handler is responsible for receiving the user’s input and writing it to the socket\*

* The thread created reads the user’s input.
* If the command is byeand the current user is the Server then the handler writes bye in the socket and exits. Else the socket in sockt is updated to the server sockt initially established.
* If the command is an initiate command the thread sets up a server for the requested user to connect to when they establish a socket. The program blocks on the accept command.
* The following code is used to make sure that multiple clients maybe coupled concurrently.

while (bind(sockfd, (const struct sockaddr \*) &servaddr,

sizeof(servaddr)) < 0) {

perror("bind failed");

servaddr.sin\_port = htons(++port2);}

* When the socket finally binds to a specific port, that port is relayed to the client to connect through.
* Else the user’s input is relayed to the socket.

### Server

Main:

* The server main establishes a socket, sets up a server and blocks on the accept command.
* When the server accepts a new connection the server initiates a connection handler thread.
* When there are no new connections the server closes.

Connection\_handler:  
\*The connection handler is responsible for responding to the clients’ requests.\*

* Initially the server expects a login command before it allows the user to issue any new commands.
* Each user needs to have a unique username, a criteria which is evaluated using a users set. If the username the client wishes to use is already present in the set then the server fails the login.
* If a list command is issued then the server responds to the user with the rest of the remaining users connected to the server. This is achieved by creating a new set diff that is a copy of the users set and erasing the current user from it. Then the diff set is forwarded to the client. The user write access is ensured by using a mutex.
* If the command is an initiate command then the handoler extracts the requested username as well as the connection port. Then the handler forwards to the requested user, whose socket is retrieved through a global map, the port to which to connect in order to establish communication beween the two clients.
* If the command is bye then the server closes the socket communication.
* If the command has any other format then the server responds that this is an unknown command and awaits for a new one to be issued.

# Problems Encountered:

Durig development we initially chose to implement the Client<->Client communication using the server as an intermediate, without any direct communication between the two Clients. In order to achieve independent Reading and Writing we implemented two threads in the Client which we kept and two threads in the Server code (one to read from one client and write to the other and vice versa). That created a very confused and unclear code that proved very difficult to debug.

After clear guidelines were provided since the original guidelines incinuated that the server should act as an intermediate which as proved was not the case we changed the implementation to achieve a direct Client<->Client communication.

This was achieved by implementing a secondary server in the initiator client and sending a corresponding message to the initiate client who acts as a client connecting to the server created.

Another problem was the blocking of the secondary port used for the Client<->Client connection did not allow multiple chat terminal to be connected but a single pair. We solved that issue by implementing a system that binds to sequantial ports that offset from the initial secondary port until it suceeds to bind and forwards the port number to the initiate client through the server.

# Runtime Examples:

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated