Code was referenced from the example Multi Threaded C server/client (For multithreading implementation), as well as the lecture UDP/TCP server/client (Primarily for setup of sockets).

Data Structures:

There are three main data structures used for the chat room: a linked list of AUTH structs which is loaded from the credentials.txt file and used to authenticate clients, a linked list of USER structs which is instantiated upon user login and stores session information used in dealing with the commands, and a linked list of MES which is used to combine message content with required metadata.

AUTH struct contains:

- Username
- Password
- Login attempts
- Next AUTH struct

USER struct contains:

- Username
- Login time
- UDP port
- UDP address
- Next USER struct

MES struct contains:

- message contents. - Fixed character array
- Send_id - User who sent the message
- Recv_id - User who receives the message
- Timestamp - Timestamp in which message sent/ modified
- Modified_B - Boolean value whether message has been edited
- Next MES struct

Operation Overview

The Client

The client is responsible for receiving the messages from the server and displaying it to the end user. In order to facilitate the user input and displaying messages from the server a multi-threaded approach was adopted - with one thread responsible for capturing user input and sending it to the server, and another used to capture information sent from the server to the client.

The client receives the message in its entirety for display onto the terminal, and will respond to the message as prompted by the server. Invalid send requests would be handled by the server.

Aside from displaying messages from the server onto the terminal, the only other logic would be that if the input to the server is OUT then the client will send a logout request to the server and then terminate. A limitation of the way it is implemented is, since the client doesn't store state, any replies to the server consisting of just "OUT" would result in the client exiting. A potential fix to this would be to store the state of the client ie. a boolean for whether the client is in "command mode" rather than authentication.

The Server

The server is responsible for the logic of handling the messages and communication between clients. Each new client that attempts to login will be assigned a thread to handle the entirety of the communication between the server and the client. This enables multiple clients to use the server simultaneously, although may introduce synchronisation problems in accessing the shared data structures of the server. This complication was acceptable as it is important to support multiple clients.

The operational overview of the server is as follows:

- Server sends requests for the username and password - if correct, setup USER struct and continue to command handler.
    - Unique usernames are treated as a sign up request.
    - Incorrect username and password combinations will increase the failed attempt counter (per account) and the server will attempt to request for username and passwords again.
    - On specified failed_login_attempts server will wait 10 secs before processing input.
- Command handler is a loop that waits for a command and calls the correct handler for the command.
    - If the command is invalid. It will notify clients to request a new command.
- Commands for message, used to the MES list to complete.
- ATU command uses the USERS list to complete.
- OUT command closes the thread.

Extension:

A proper sign up function would be important to implement. Currently the server considers every unique username input to correspond to a sign up request. This could be problematic if a user enters the username incorrectly and the server signs the user in with a new account that is not their own.