Project 4 OS report
Ian Spiegel, Chris Smith
12/1/2019

In this project, we implemented the main server to display an up-to-date list of connected clients when a client connects or disconnects, the main client to specify user name when connecting to the server (and maintain that user name until it disconnects), main server allowing multiple chat rooms to run at the same time, client is able to make new rooms or connect to an existing one, if the client doesn't specify room number or the new keyword, then it'll retrieve a list of rooms available currently then, and colors are used for each new username to help identify people in the chat room.

Whenever a new client connects to the server, the information is sent to the server and stored in a buffer that lists the server users by username. This is how we can easily keep track of who is on the server, and whenever a client disconnects they are removed from that buffer. This information is sent via the SENDIT function implemented within the client side. The server side will take in that data and organize it to the right location.

Using information from the full client and server examples given to us, we were able to create a method that allows multiple clients to connect to a singular server. Essentially, when a new client connects, the master socket will activate and a new forward will be open for that client. We then store that forward in our client buffer and in a following iteration, we add it to a separate reading forward buffer to update on client activities.

Whenever a new client comes into the server it will be able to give a username for themselves, this was a simple scan from the terminal side of a client, stored in a buffer, and shipped to the server. This seems to have a slight issue though we noticed, when the first client tries to login to the server, they sometimes do not receive an option to place in their username. But always clients after the first will have the option no matter what.

Using the command "new" the client is allowed to create a new chat room rather than only select from the already pre-existing rooms. The server can also have multiple chat rooms running at once due to our coding in parallel. Also, placing a room number after the IP address of the client server connection, the client can instantly join a room without having to be asked by the server first. Only placing the IP address will work fine as well though. Our concurrent programming in our code allows for these chat rooms to co-exists simultaneously without crashing the server or client(s). The max amount is 10 rooms.

Our program in the server detects when there are no rooms available and will automatically force the "new" function on the client if there are no rooms. If there is a room, then the user can decide which room to choose whether it be new, random, or a pre-existing one. If the client doesn't specify a room number, it will receive a list from the server telling him or her what is available to choose from, if there is anything available to choose from that is. Each room has its own fork for sending and receiving, and we store the data that a room exists (with a number)

inside a buffer. We check to see if there is anything in this room buffer, if not, we make a new room, if there is, then we display it for the client to view and choose from.

Finally, using a package found online that allows the display of colors for certain text in a terminal, we used that package to randomly generate certain colors onto the text of different usernames in a room. This color stays the same for all texts from a certain person so it's easy to identify who is talking in a conversation. The rand() function allows the color selection for each person to be random, so it's a surprise what color you will get for yourself and fellow clients in your room.

(We did not do the extra credit)