Student no.221405356

Network Systems 3

Laboratory 3

Tshibanda Emmanuel Mukendi

In the Client Python File

This code creates a client-side network connection with a server at IP address "192.168.1.235" and port 9090 using the TCP protocol. We see this by the parameter sock.stream. Additionally, it builds a GUI interface for user interaction with the program using the Tkinter library.

Two basic elements make up the GUI interface: a text box for displaying chat messages and an entry box where users may type messages to send. Font, colour, and size properties are set for the text box and entry box.

The user is then prompted for their username, which is sent to the server via the established socket connection, by the code.

A second thread is used by the "receiving" function, which monitors incoming messages from the server. The sender's username and message are then extracted from the received data and shown in the GUI text area.

When the user presses the enter key or clicks the send button, the "sending" function is initiated. Over the established socket connection, it transmits the entered message to the server.

A program shutdown event handler is also included in the code, which removes the GUI interface and shuts the socket connection.

Overall, this code produces a straightforward chat client that gives users a graphical user interface for sending and receiving messages over a network.

In the Server Python File

The software uses the TCP/IP protocol to build a straightforward chatroom server. It is written in Python. The'socket' and'select' modules from Python's standard library are used by the server.

The program starts by importing the necessary modules and setting up a few constants, such as the header length, server IP address, and port number.

Using the 'listen()' method, the server socket is created, assigned an IP address and port number, and then made to watch for incoming connections. To enable the reuse of the local address, the socket option is configured using the'setsockopt()' function.

.

The'serverSocket' object is used to initialize the'socketsList' list, and a dictionary called 'clients' is also constructed to keep track of the clients that are currently connected.

'receive_message' is a specified function that receives messages from a client and returns a dictionary that contains the message header and the message contents.

The'select()' function, which stops until at least one of the connections in'socketsList' contains incoming data, is used by the program's main loop to monitor incoming messages. The'read_sockets', 'write_sockets', and 'exception_sockets' lists of sockets are the three lists that the method returns.

The program then iterates over the list of open "read_sockets" and processes the incoming data according on whether it is a new connection request, a message from a client that is currently connected, or an error. The client socket is accepted and the client is added to the "clients" dictionary if it is a new connection request.

If it is a message from a connected client, the message is broadcast to all the other connected clients. If the data is False, that means the client has left the chatroom. In this case, the client is removed from the `socketsList` and the `clients` dictionary.

Finally, any exceptions that occur while processing the sockets are handled by removing the socket from the `socketsList` and the `clients` dictionary.