Network Socket Specifications

The communication protocol relies on JSON packets to ensure concise and structured data exchange. Each packet consists of two main parts:

1. Packet Length Field: A 2-byte length field, sent in big-endian format, specifying the length of the following JSON packet. This ensures that the entire packet is read correctly.
2. JSON Packet: This packet contains the following three fields:
    - 'msg': The incoming message, if any.
    - 'nick': The nickname of the person who sent the message (set to None upon connection and changed upon approval). The server reserves the nickname 'SERVER'.
    - 'proto': The protocol used for communication, determining the nature of the message exchange.
    - 'connect': Used by the server to acknowledge the client's connection and send the message 'HELLO'.
    - 'verify': Used by the client to inform the server that it is in the nickname verification step, with the nickname in the 'msg' field.
    - 'broadcast': Used by the client to indicate a desire to broadcast a message to all other connected clients.
    - 'goodbye': Used by the client to notify the server of its intention to disconnect.
    - 'shutdown': Used by the server to inform connected clients of an impending server shutdown, with the 'msg' field containing a countdown message.

Reading

To read a packet:

1. Read the first two bytes, convert them from big-endian if necessary.
2. Read the number of bytes specified by the length field to obtain a JSON object.
3. Decode the JSON object, which can be used as a dictionary to access 'msg', 'nick', and 'proto' fields.

Sending

To send a packet:

1. Create a JSON packet with 'msg', 'nick', and 'proto' fields filled with data.
2. Determine the length of the JSON object in bytes.
3. Append the length as a 2-byte big-endian indicator to the beginning of the JSON object.
4. Send the packet to the socket.

Chat Protocol Specifications (Client Side)

The chat protocol for the client side involves the following steps:

1. Connection Setup:
    - Open a socket based on command-line information.
    - Connect to the server using command-line information.
    - Upon connection, receive a packet from the server with 'msg' = 'HELLO' and 'proto' = 'connect'.

2. Nickname Verification Loop:
    - Prompt the user to select a unique nickname for the chat room.
    - Send a packet to the server with 'proto' = 'verify' and 'msg' containing the chosen nickname.
    - Handle server responses:
    - If the nickname is accepted, receive a packet with 'msg' = 'READY' and 'proto' = 'ready', indicating that the client can start chatting.
    - If the nickname is not accepted, receive a packet with 'msg' = 'RETRY' and 'proto' = 'retry', prompting the user to choose another nickname.
    - Once an accepted nickname is established, move on to the chatroom.
3. Chatroom:
    - Continuously prompt the user for input and check for incoming messages from other clients or the server.
    - To send a message:
    - Input a message.
    - Pack it into a JSON packet with 'msg', 'nick', and 'proto' fields, with 'proto' = 'broadcast'.
    - Append the JSON object's length in bytes to the packet.
    - Send the packet to the server.
    - The server broadcasts the message to all connected clients, including the sender.
4. Receiving Messages:
    - Receive messages by reading packets.
    - Unpack the JSON packet to extract 'nick', 'msg', and 'proto' fields.
    - Supported protocols are 'broadcast' (messages from other clients or the server) and 'shutdown' (server's intent to shut down).
5. Exiting the Client:
    - The user can exit the client by pressing Ctrl+C. The client sends 'msg' = 'BYE' and 'proto' = 'goodbye' to signal its intention to disconnect.
    - Close the socket, handle cleanup, and exit the client.

Chat Protocol Specifications (Server Side)

The server's protocol involves managing client connections and handling communication:

1. Connection Setup:
    - Open a socket based on command-line information.
    - Bind to the specified information and start listening for incoming connections.
    - Use an asynchronous I/O mechanism (e.g., select) to handle incoming connections.
2. Handling New Connections:
    - When a new connection is established, the server accepts it, sends a greeting, and monitors for client activity.
    - Send a packet to the client with 'nick' = 'SERVER', 'msg' = 'HELLO', and 'proto' = 'connect'.
3. Handling Messages:
    - Use the selected I/O mechanism to poll sockets for input.
    - Depending on the protocol in the received packet, take appropriate actions:
    - 'verify': Check if the chosen nickname is unique and responds with READY or RETRY.
    - 'broadcast': Relay messages to all connected clients, log it, and continue I/O handling.
    - 'goodbye': Handle client disconnect by broadcasting a message, removing the connection and nickname, and closing the socket.
4. Server Shutdown:
    - To shut down the server gracefully, broadcast a warning message to all clients with 'msg' = "Server Shutting down in x seconds" and 'proto' = 'shutdown'.