COMP26120
Lab Exercise 2 - Report

Feroz Hassan

A More Robust Messaging System

1 Protocol Design

The Messaging System protocol allows clients to connect to the server and communicate with each other. Design decisions have been made to ensure user-friendliness, usability, security, and robustness of the system.

Command	Description
1. name <your_name>	Set a screen name
2. sendto <user> <message>	Send a message to a particular user
3. sendall <message>	Send a message to everyone
4. list	List all connected (active) users
5. help	Show a list of all commands
6. quit	Close connection with server

1.1 Commands

1. The `name` command is used to set a screen name for the client. This command takes a single parameter which is the name the client wishes to use. The server checks if the name is available or not. If the name is already taken, the server sends an error message to the client asking them to choose a different name. If the name is available, the server assigns it to the client and adds the client's socket to the `users` dictionary.

   If a client wants rename themselves, they may use the same command

2. The `sendto` command is used to send a message to a particular user. This command takes two parameters: the recipient's name and the message to be sent. If the recipient's name is not found in the `users` dictionary, the server sends an error message to the client. Otherwise, the server sends the message to the recipient's socket.

3. The `sendall` command is used to send a message to all connected users except the sender. This command takes a single parameter which is the

message to be sent. The server iterates over all sockets in the `users` dictionary and sends the message to each socket except the sender's socket.

4. The `list` command is used to list all active users. This command takes no parameters. The server iterates over all entries in the `users` dictionary and sends a list of all active users to the client.
5. The `help` command is used to display a list of all available commands
6. The `quit` command is used to close the connection. If a user wants to disconnect then a message is sent to the server to requesting to disconnect and the client should will exit after only after receiving confirmation from the server.

1.2 Error Handling

In the server implementation, error handling is implemented to ensure a robust and user-friendly experience. Some key error-handling strategies include:

Checking for invalid screen names: When a user tries to set a screen name, the server checks if it meets the specified criteria (i.e., containing only letters, numbers, and underscores, and having a length between 3 and 20 characters). If the screen name is invalid, the server sends a corresponding error message to the client, which is then displayed to the user.

Duplicate screen names: If a user attempts to register a screen name that is already taken, the server sends an error message to the client which then informs the user that the chosen screen name is unavailable, prompting them to choose a different one.

Catching exceptions: The main loop for handling user input is wrapped in a try-except block to catch exceptions that might arise during the execution of the program. Two specific exceptions are handled: **KeyboardInterrupt** and **OSError**. In the event of a KeyboardInterrupt, the client sends a "quit" message to the server, indicating that the user has requested to terminate the connection. If an OSError occurs, the client displays an error message, notifying the user that the server might be down and asking them to check its status.

1.3 User-friendliness

In the given server implementation, user-friendliness is an essential aspect of the design. The server offers a simple and intuitive command structure, allowing users to easily navigate and engage with the system.

Furthermore, the server validates user inputs, ensuring that invalid commands or improperly formatted messages are not processed, and instead, informative error messages are returned to the user. By offering a clear and comprehensive set of commands coupled with helpful feedback, this server implementation enhances user-friendliness, making it easier for clients to interact and communicate with each other.

2 Pseudocode

```
function onStart()
    Display message to acknowledge server started
    Initialise server side variables like client count and users dictionary

function onStop()
    Display message to acknowledge server stopped

function onConnect(socket)
    Increment client count
    Initialse client specific variables inside socket object
    Display welcome message including commands, usage etc.

function onMessage(socket, message)
    Extract command and parameters from message
    if command is "name" then
        Validate name to match criteria
        if name is available then
            Assign name to client and add client to users dictionary
            Send confirmation message to client
        if name is not available then
            Send error message to client
            Set client's name and update users dictionary
    else if command is "sendall" then
        Ensure user is registered
        Validate message to check it's not blank
        Send message to all connected users
    else if command is "sendto" then
        Ensure user is registered
        Validate message to check it's not blank
        Get recipient's socket from users dictionary using parameter as key
        Send message to a specific user
    else if command is "list" then
        Return a list of connected users
    else
        Send "Invalid command" error to socket

    return True to signify everything is fine

function onDisconnect(socket)
    Decrement client count
    Remove socket's screen name from users dictionary
    Close socket
```