Michael Long
Gennadii Sytov
CS494 Chat Application
Spring, 2019

Table of Contents

Disclaimer

The software discussed within this document only serves as a proof of concept for demonstrating features and functionality and applying concepts learned from the course, CS494 Internet Networking Protocols. It is not designed to be secure or used regularly and additional design considerations need to be made to protect users and each system on a live connection.

Definitions:

User: an individual or program to give input.

Client: application designed for a user to connect to the server with, primarily to exist as a medium to write input and receive output.

Server: application designed to handle zero or more connections from different clients.

Administrator: an individual or program who defines the initialization of the server.

Message: a string sent over the connection of maximum length of 1024 bytes encoded with UTF-8.

Command: a message prefaced with a “/” to request an action from the server

Username: an label used by the server to identify a user stored by the server.

1. Introduction

This document describes and summarizes the functionality of a basic Internet Relay Chat protocol, (IRC), which will be referred to as “Chat Application”. Chat Application is written in Python 3.7.2. Chat Application allows any number of users to communicate via shared messages that are congregated into different “rooms”. This is handled by two programs, a server application which listens for input, and zero or more client applications which are responsible for sending input to for the server application to handle. A user is able to do the following: obtain a list of commands, send a shared message to all rooms they’re in, send a shared message to a specific room, send a private message to another user, create a new room, join a room, leave a room, list each user online, list each user in a room, list all created rooms, and disconnect from the server.

2. Setup

Chat Application’s server application obtains it’s address space from “AF_INET” from Python’s “socket” library, in which the address is represented as an IP address or domain name and a port number to connect the socket to. The transport layer is the standard Transmission Control Protocol (TCP) provided by “SOCK_STREAM” from the “socket” library. With these two characteristics, the server application is able to successfully launch and receive input from each client application. Initial testing was done in the scope of a local host (localhost) with IP of 127.0.0.1 and a port number of 1234. For the scope of this project, localhost testing is sufficient to demonstrate functionality and features. Each “message” is sent as a byte stream with maximum buffer side of 1024 bytes encoded with UTF-8. For clarity, each instance of message in the following sections refer to this definition.

3. Client

3.1 Application Details

The client application is composed of two files, “CS494_Client_Main.py” and “CS494_Client_Handler.py”. CS494_Client_Main.py acts as a controller to initiate the input thread, receiver thread, and establish the initial connection. The input thread will allow a user to input text into a console to form messages and the receiver thread will allow the client to listen for output from the server. CS494_Client_Handler is responsible for receiving output from the server, interpreting output, and the ability to repeatedly input messages.

3.2 Establishing a Connection

When the client is initialized, an IP address and port number will be requested to attempt to establish a connection. If both the IP address and port number are valid, the client will successfully connect to the server and receive a byte stream requesting a username. If either the IP address or port number are not valid, then it will attempt a connection with the bad input, and result in a rejected connection followed by a request for both a valid IP address and port number. Once the user is able to establish a connection, a request for a username will be received from the server. Each client must have a unique username per the implementation, and a client will not be allowed to feed input to the server until this has been established. [See Section 4.3]

3.3 Sending Input

The client maintains an input thread defined in CS494_Client_Handler. The client, as long as it's connected to the server, will be able to enter text into the console and send it through the socket to the server. Most input is handled and interpreted by the server. The only instance in which the client will interpret text from the console is if the user uses the "/quit" command in which the client will notify the server then close the socket. The client communicates commands to the server by prefacing the text with a forward slash: "/" [See Section 4.5 for specific use cases]. The user will be able to prompt the server to either obtain a list of commands, send a shared message to all rooms they're in, send a shared message to a specific room, send a private message to another user, create a new room, join a room, leave a room, list each user online, list each user in a room, list all created rooms, and disconnect from the server.

3.4 Receiving Output

The client maintains a receiver thread defined in CS494_Client_Handler. The client will be able to receive and interpret output from the server through a listener on the socket. Most output is echoed directly to the console in either a message from a user or an error message on input sent to the server. The client is able to interpret special messages sent from the server to perform specific functionality. In the instance of a --disconnect-- message sent from the server, the client will be notified that they have been disconnected from the server. Output received from the socket is specific and is only readily available for the intended client(s).

3.5 Impromptu Disconnections / Crash Handling

In the event of a user improperly closing the application or a loss of connection, the server will receive an exception in the next instance it deploys the message and adjust accordingly. The client is able to disconnect in any method and the server will maintain its connection with each other connected client. Each instance of a disconnection is isolated from the other clients, however each client will be notified that a user has disconnected from the server.

4. Server

The server application is composed of two files, “CS494_Server_Main.py” and “CS494_Server_Handler.py”. CS494_Server_Main.py initializes the main server thread and establishes which IP and port the server will be hosted on. CS494_Server_Handler is responsible for handling various data structures, creating new threads for each connected client, and outputting byte streams to appropriate the appropriate client(s).

4.1 Server Initialization

The server must successfully establish a binded socket with an IP address and port number before any messages can be received from a client. When the server is first boot, a server administrator will need to specify both the IP address and port number. If the server is able to successfully bind to the socket, then each client will be able to send input to the server. If the server is unable to bind to the socket, then the server will request an IP address and port number again until a connection is able to be made. For the scope of this project, the client will be hosted as a “localhost” (IP 127.0.0.1) on port 1234. The server console will notify the administrator when it has been successfully launched through the console.

4.2 Data Types

The server maintains data structures to hold basic routing information and information on each room and where to send data based on input received. Each client is stored in a “dictionary” [See Reference 1] called “users” in which a “username” defined by the user and the corresponding value is a reference to the respective “conn” information of the corresponding user’s client. The username will allow the server to easily identify each client and the conn will allow the server to have the forwarding information necessary to send messages through the socket to the appropriate client. Each room is stored in a dictionary called “rooms”, in which a “room name” is the key and the corresponding value is a list of users who are in the room. Rooms are created by users and the list of users change based on when a user joins a room, leaves a room, or their client is disconnected from the server.

4.3 Connecting a Client

Given the server has successfully initialized, the main server thread will deploy a listener to begin seeking client connections from the socket. In the event that a client is able to successfully connect to the binded socket, the server will send a welcome message followed by a prompt for the user to input a unique username. If a username is not unique, the server will continually request a new username until a unique username is specified. Empty usernames are forbidden, and will result in the server disconnecting the client and closing the corresponding thread. This is to prevent bogus information from being passed to the server. The username will immediately be added to the room ‘General’ and the server will send the corresponding client a message introducing the functionality.

4.4 Receiving Input and Commands from a Client

Once a connection has been successfully established, the thread corresponding to the connected user will be open for input .Any message not prefaced with a "/" will be reformatted and sent to each client whose username is in a room with the client that sent the message. A client will be able to send specific "command" to the server by prefacing text with a "/". Messages sent beginning with a "/" will be compared to a series of valid commands with a corresponding syntax. An invalid command will result in the server sending an error message to the corresponding client. The server is able to process the following commands:

/? -- list all commands available
/create room_name -- creates a new room
/join room_name room_name . . . room_name -- user joins each room_name
/leave room_name -- leaves room_name
/users -- list all users
/users room_name -- list all users in room_name
/list -- list all rooms
/s room_name . . . room_name message -- send a message to all users in each room_name
/w username message -- send a private message to a user
/disconnect -- disconnect from the server

4.5 Creating a Room

"/create room_name"

The server will be requested to create a new room labeled as room_name. If the room with the corresponding room_name does not exist, then the server will create a new dictionary entry in rooms. If a room with the corresponding room_name has already been created, then the room will not be created and the client will receive an error message.

4.6 Joining (a) Room(s)

"/join room_name room_name . . . room_name"

The server will be requested to allow a user to join one or more rooms. A room will not allow multiple instances of the same username to prevent erroneous behaviors. This command runs to completion and will attempt to resolve each room_name specified received from the client. If each room_name exists in the rooms data structure then the username of the corresponding client will be added to each room. If one or more room_name is not defined in rooms, then the server will send an error message for each invalid room. If the client's username is in one or more of the rooms, then the server will send an error message for each room the user is in. If no room_name is specified in the command, then the server will treat the command as invalid and send the client an error message.

4.7 Leaving a Room

“/leave room_name”

The server will be requested to remove a user from a room. If the username of the corresponding client is in the room, the username will be removed from username list in the corresponding rooms entry. If the username is not in the room or the room does not exist, the server will send an error message to the corresponding client.

4.8 Special Messaging

“/s room_name . . . room_name message”

The server will be requested to send a specific message to each client whose username is in each listed room. The command will parse from left to right until the room_name does not exist, then will send the invalid room_name and the rest of the message to each client of each username in each room. If no room matches or is empty then no message will be sent.

“/w username message”

The server will be requested to send a message to the client of the corresponding username. If the username does not belong to any client, then no message will be sent. If the username is empty then no message will be sent.

4.9 Showing Users, Rooms, and Commands

“/?”

The server will be requested to send a list of all valid commands. This information is received from the “commands” list which contains a series of definitions for the client to use. [See Section 4.4]

“/list”

The server will be requested to list all rooms that have been created. This information is received from the rooms dictionary in which each key is sent to the client.

“/users”

The server will be requested to list all users that are currently connected to the server. This information is received from the users dictionary in which each key is sent to the client.

“/users room_name”

The server will be requested to list all users that are currently connected to the server. This information is received from the rooms dictionary in which each value with the associated room_name is sent to the client.

4.10 Disconnect

“/disconnect”

The server will be requested to close the connection of the address of the username corresponding to the client. The server will disconnect the username, delete the username from each room they are in, delete the username, then close the corresponding thread. This allows the username to be used by other incoming usernames. Once a client has been disconnected, they will lose all data associated with the username.

4.11 Disconnection Handling

The server is able to handle any client disconnection and has a different response based on how the disconnection occurs. In the event that a user types “/quit” into their client, the client will close their connection to the server, the next instance of the the corresponding thread maintaining the connection executes, it will catch an exception and delete the username from each room they are in, delete the username, then close the corresponding thread. In the event that a client sends a “/disconnect” command to the server, it will follow the functionality previously defined [See section 4.9]. In the event that a client is closed or loses connection before a username is specified, then the server will close the corresponding thread opened. In the event that a client loses connection to the server after they have specified a username, the next instance of the the corresponding thread maintaining the connection executes, it will catch an exception and delete the username from each room they are in, delete the username, then close the corresponding thread. In the event that the Server crashes, each client connected on the binded socket will be disconnected and will inform the user that a disconnection has occurred.

5. Features

The following features have been implemented per the “CS 494/594 Programming Project Specification Spring 2019” and “IRC Grading Sheet” [See References 2 and 3]. As such, the following features have been implemented:

2 Server process exists [Section 4]
3 A client can connect to the server [Section 3.2]
4 Client can create a room using /create room_name command [Section 4.5]
5 Client can list all rooms using /list command [Section 4.8]
6 Client can join a room using /join room_name command [Section 4.5]
7 Client can leave a room using /leave room_name command [Section 4.6]
8 Client can list members of a room using /users room_name command [Section 4.8]
9 Multiple clients can connect to a server [Section 3.3 and 4.3]
10 Client can send messages to a room [Section 4.7]
11 Client can join multiple (selected) rooms [Section 4.5]
12 Client can send distinct messages to multiple (selected) rooms [Section 4.7]
13 Client can disconnect from a server using /quit command [Section 3.4]
14 Server can disconnect from clients using /disconnect command [Section 4.9]
15 Server can gracefully handle client crashes, [Section 4.9]
16 Client can gracefully handle server crashes, [Section 3.4 and 4.9]

6. Extra Features

6.1 Private messaging

Users can send private messages to each other using “/w username message” command. [See Section 4.7]

7. Conclusion and Considerations

This document outlines and specifies the features and usage of Chat Application as some number of users are able to communicate through different mediums either in a shared scope, to specific rooms, or to one-another. From this basic implementation additional functionality can be added such as the ability to do file transfer, registered usernames protected with some password, and additional logging features on the server application to maintain records of offline users, users messages, and chat histories. With this system there is little consideration for security or data protection, as it is outside the scope of this project’s main features.

References:

1. Python 3.7.2 Reference Manual -- Dictionary:
https://docs.python.org/3/tutorial/datastructures.html

2. CS 494/594 Programming Project Specification Spring 2019, Dr. Nirupama Bulusu, 06/2019

3. IRC Grading Sheet, Dr. Nirupama Bulusu, 06/2019