```
Internet Draft                                              Huanhua Su
CS494                                         Portland State University
RFC Documents for IRC                                       20/22/2020
```

Internet Relay Chat Protocol

Status of this Memo

For formal document, please refer to the IRC RFC 1459:
https://tools.ietf.org/html/rfc1459

Abstract

This is a project assigned by course CS494 Network Protocol. Its goal is to develop and demonstrate a working IRC protocol and implementing an IRC server and client in any language that supporting socket API. Specifically, this program is written in Python. Recalled that IRC is a message-based protocol that allows many IRC clients to communicate together via by connecting to a running IRC server through socket connection. The implementation of the this IRC protocol support multiple chatting functionalities such as chatting within a chatting room, sending private message, file sharing, etc.

Table of Contents

1. Introduction

This specification describes a simple Internet Relay Chat (IRC) protocol by which clients can communicate with each other. This system employs a central server which act as an intermediary machine that help transmitting messages for connected users.

Users/clients can create, list, join, leave rooms, and any message sent to that room will be forwarded to all users currently joined to that room.

Users can also send private messages directly to other users, can also send secure messages to another user, and can also transfer file to a specific user.

2. Basic Information

All communication described in this protocol takes place over TCP/IP, with the server listening for connections on port 29. Clients connect to this port and maintain this persistent connection to the server. The client can send messages to the server over this open channel, and the server can reply via the same. This messaging protocol is inherently asynchronous - the client is free to send messages to the server at any time, and the server may asynchronously send messages back to the client.

The server MAY choose to allow only a finite number of users, depending on the implementation and resources of the host system. Right now the server implementation only allow 5 client connections for experiment purpose.

3. Message Usage

3.1. Message Format

The messages command that is used in this protocol are all in 4 letters word as “COMD msg msg msg”. The letters can be lowercase or uppercase. So for example, the message command to create a room is ROOM, to join a room is JOIN, to leave a room is LEVE, etc. Note that these 4 letters are taken from the original word of message. Message commands may or may not have other parameters along with it, depends on the usage of that specific message command.

3.2. Error codes

There are couple error codes that come back from server after client send a message out. Its purpose is to inform client of a specific situation such as message command is executed successfully, or there are other errors to show that the command is failed to execute. These error code are in 3 digit format, such as 200 means OK, 300 means something already exist (such as a room already exist when a user create a room using a room name that is already been used).

3.3. Response Message

The response message from server having a format: <error code> <message>, which the error code will be analyzed and perform specific operations by the client program and it will only show the response message to the clients.

4. Client Messages

4.1. Client Connect

When client connect to the server, the client program will prompt for a username that use to specify a connected client. It is now be done on the client program by issuing a message command: USER <username>, taken the client input as the username. This message can only be used once during the time on connection, and client will be prompted repeatedly until he/she input a unique username for the connection. Clients can’t use this message after get into the chatting system hosting by server. If client somehow get into the system without give a username, he/she is not allowed to execute useful commands to the server and will receive response as:

600 <error message>

4.2. Help message

Command: HELP

This command takes no parameter, and will print out a list of available commands for user to reder.

Response:
200 <message>: An OK message.

4.3. Listing Rooms

Command: LIRO

This command takes no parameter, and will print out a list of available rooms by room name.

Response:
200 <message>: An OK message.

600 <error message>: Client is not valid.

4.4. Listing Members Of A Room

Command: LIME <room name>

This command takes 1 parameter, a name of room that exist currently. and will print out a list current members in this room.

Example:
lime myroom: listing members in a room called “myroom”.

Response:
200 <message>: An OK message.
300 <error message>: Error, room does not exist.
400 <error message>: Error, invalid room name (empty space).
600 <error message>: Client is not valid.

4.5. Creating Rooms

Command: ROOM <room name>

This command takes 1 parameter, and will create a room that is now not exist in the system.

Example:
room myroom: Create a room called “myroom”.

Response:
200 <message>: An OK message.
300 <error message>: Error, room already exist.
400 <error message>: Error, invalid room name (empty space).
600 <error message>: Client is not valid.

4.6. Joining Rooms

Command: JOIN <room name>

This command takes 1 parameter, and will add current client to the target room.

Example:
join myroom: Join a room called “myroom”.

Response:
200 <message>: An OK message.
300 <error message>: Error, room doesn’t exist, or current client already in the target room.
400 <error message>: Error, invalid room name (empty space).
600 <error message>: Client is not valid.

4.7. Leaving a Room

Command: LEVE <room name>

This command takes 1 parameter, and will remove current client from the target room.

Example:
leve myroom: Leave a room called "myroom".

Response:
200 <message>: An OK message.
300 <error message>: Error, room doesn't exist, or current client is not in the target room.
400 <error message>: Error, invalid room name (empty space).
600 <error message>: Client is not valid.

4.8. Sending Messages To A Room

Command: SEND <room name> <message>

This command takes 2 parameters, then sent a message to all members in a target room with the current client as the sender.

Example:
send myroom hello everyone!!: Send a message "hello everyone!!" to a room called "myroom".

Response:
200 <message>: An OK message.
300 <error message>: Error, room doesn't exist, or current client is not in the target room.
400 <error message>: Error, invalid parameters.
600 <error message>: Client is not valid.

4.9. Private Message

Command: PRIV <username> <message>

This command takes 2 parameters, then forward a message to a target user with the current client as the sender.

Example:
priv Bob hello bob!: Send a private message "hello bob!" to a user called "Bob".

Response:
200 <message>: An OK message.
300 <error message>: Error, target user doesn't exist.

400 <error message>: Error, invalid parameters.
600 <error message>: Client is not valid.

4.10. Using A Key

Command: USEK <keyword>

This command takes 1 parameter, then store the keyword as a key for encrypting/decrypting when using secure message.

Example:
usek abc: use keyword "abc" as a key.

Response:
No response from server, this is a client-side command.

4.11. Secure Message

Command: SECU <username> <message>

This command takes 2 parameters, and will send a encrypted message to a target user. The encryption and decryption are done on client side so server can only see the encrypted version of message.

Example:
secu bob this is secret: Send an encrypted message "this is secret" to a target user called "bob".

Response:
200 <message>: An OK message.
300 <error message>: Error, target user doesn't exist.
400 <error message>: Error, invalid parameters.
600 <error message>: Client is not valid.
700 <sender> <message>: For receiver, inform that this is a secure message for further operations on client program.

4.12. File Transfer

Command: FILE <username> <filename>

This command takes 2 parameters, and will send a file from current user's current folder to a target user.

Example:
file bob file.txt: Send a file with path name "file.txt" to a target user called "bob".

Response:
100: Server tells sender it is now ready to receive file.
101 <message>: File transfer failed.

102 <filename> <file size> <message>: For receiver, inform that this a file transferring for further operations on client program.
200 <message>: An OK message.
300 <error message>: Error, target user doesn’t exist.
400 <error message>: Error, invalid parameters.
600 <error message>: Client is not valid.

4.13. Quit

Command: QUIT

This command takes no parameter, and will exit the client program.

Response:
No response from server, this is a client-side command.

5. Server Messages

5.1. Unknown Command

When server receive any unknown commands that are not listed on the help message, it will respond a message as:

500 <error message>

5.2. Server Quit

To quit the server when running the server in interactive CLI properly, enter ‘q’ as ‘quit’ and hit Enter, then the server program will quit.

When server quits or crashes, clients will receive a message with no data. In this case, all clients will perform disconnection on their own program.

6. Error Handling

Both server and client will detect when the socket connection linking them is terminated. If the server detects that the client connection has been lost, the server will remove the client information from the system including all room information they are joining. If the client detects that the connection to the server has been lost, it also will disconnected the connection.

Upon any clients disconnect from server, server will notify other clients online about this event. Also, when server disconnect, all clients will terminate the client program.

7. Cloud Server

There is a cloud server running as an AWS EC2 instance for this project. The server can be reached by “irc.karanokara.com” at port 29.

This cloud server will be expired on 03/23/2020 midnight as the end of the term.

8. Security Considerations

Messages sent using this system MAY have no protection against inspection, tampering or outright forgery. The server can sees all messages that are sent through the use of this service UNLESS client use the Secure Message, in which case the message will be encrypted using a client’s given key, and server can only see the encrypted version of sent message.

9. Acknowledgments

This document was prepared using Google Doc by Huanhua Su.

Email: Huanhua@pdx.edu