Gabrielle George, Christopher Lu, Sean Sweeney

Intro

Herein we describe a basic version of a well-known protocol - the Internet Relay Chat protocol. Users are able to communicate with a server which facilitates messaging between various users. A server hosts some number of “room” abstractions that users may join. A user who is part of a room receives all those messages sent by other users in the room and, likewise, all other users in that room receive messages sent by the user. The server, then, acts as a relay between the various users with which it is communicating.

We define some basic structures used by users and servers to communicate with each other, namely the various sorts of messages that might be sent from a user to a server and from a server to a user. We go on to lay out some basic software design principles we plan to use in implementing this protocol. Finally, we conclude with a quick discussion of the sorts of errors that might be encountered in the course of client-server communication and how those errors are handled.

Servers

To quote RFC 1459, Servers are the backbone of IRC. This project attempts its best to emulate the behavior of a server at a relatively small scale. Our servers provide “a point to which clients may connect to talk to each other”, but not a means for other servers to connect to in order to form a true IRC network with other servers sending and receiving information from each other.

Our servers are intended to allow a limited number of client applications to connect to the server. With a number of features implemented by the team, servers broadcast information to user clients, and hold room information (including room membership).

Client

Clients are application instances we implemented that connect to the server through the Python socket library. Clients are distinguished by unique numerical IDs. These IDs are assigned by their integer return value of the socket file descriptor these are unique identifiers at the scale of our protocol.

Rooms

Similar to Channels in the IRC specification, rooms are a named group consisting of two or more clients. Rooms are created when the first client joins them. Since the deletion of rooms or client room membership is not in the scope of this project so rooms and their members persist until the server loop terminates.

There exists only one type of room in this protocol, all users are capable of viewing, joining, and messaging all created rooms. Users can create rooms with names of their designation.

Protocol Specification

Summary

The protocol as described below is for use for both server to client as well as client to server; connections from both sides are regarded as equally trustworthy. For certain valid commands, the client will receive a reply, though it does not wait on that reply, for other commands the server will not send any reply. Based upon the intended design, server response is quick, however, client-to-server communication is functionally asynchronous and dependent upon server response.

Message Format

Every message is modeled as a string of bytes. The first word (space-separated substring) of the string is the header of the message, every word thereafter constitutes the body of the message. The header of the message is an all-capital letter string denoting which sort of message it is. The body of the message varies from message type to message type with some messages able to have quite a few more words and others only empty bodies.

The length of a message may not exceed 1024

Message Types

Servers and clients send each other messages, and replies which may or may not generate responses clients will see.

Messages are structured as UTF-8 character strings delimited by spaces. Commands and the following parameters are separated by one ASCII character. The first word preceded by a slash (/) is a command, and the following word is the body of the command. Invalid commands error out as invalid. Messages are limited to 1024-byte strings.

Quit:

Parameters: None
A client session is ended with a quit message and their information is deleted from the room they were in as well as the server.
Example: > /quit

Join:

Example: >/join room1
Parameters: name of the room
If the room does not exist it is created and added to the server.

Leave:

Example: >/leave room1
The user is moved out of the room and their membership from the room is removed.

Room membership:

```
Example: >/listrooms
Available Rooms:
room1
```

The user that sends this command to the server receives a response with a list of available rooms.

Message Room:

```
>/msg Hello There
sending to room: Hello There
>/msg #room1 : Hello There Again
sending to room: #room1 : Hello There Again
```

Users are capable of messaging rooms specifically. If a message is detected with no body (which would be a room name) its message is sent to the room the user occupies. Users can be removed from a server if they type words the server blocks.

Private Message:

```
>/dm person hello
```

Users are capable of messaging specific individuals connected to the room by

References:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#successful_responses
https://datatracker.ietf.org/doc/html/rfc1459
https://docs.python.org/3/library/socket.html