Internet Relay Chat Class Project

Abstract

In this project, we will implement the internet relay chat.The programming language in use is python. The central focus on the chat functionality is because this is our first term, so we want to start small. The ambitious future of this project will be chat and voice functionality with secure messaging.

Team Members

- Arpankumar Rajpurohit - arpan2@pdx.edu
- Krishna Sai Panthala - panthala@pdx.edu
- Siva Sai Kumar Paritala - siva2@pdx.edu

Table of Contents

1. Introduction

This specification describes a simple Game with chat functionality (IRC) protocol by which clients can communicate with each other. This system employs a central server which "relays" messages that are sent to it to other connected users. Users can join rooms, create rooms which are groups of users that are subscribed to the same message stream. Any message sent to that room is forwarded to all users currently joined to that room.

2. Basic Information

All communication described in this protocol takes place over TCP/IP, with the server listening for connections on port 31425. Clients connect to this port and maintain this persistent connection to the server. The client can send messages and requests 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. Both the server and client may terminate the connection at any time for any reason. They may choose to send an error message to the other party informing them of the reason for connection termination. The server may choose to allow only a finite number of users and rooms, depending on the implementation and resources of the host system. For the current project, the client connection limit is 6.

3. Message Infrastructure

Client sends the commands to perform, in format of command argument1 argument2 where command is a four letter word which can be in any format (uppercase, lowercase or combination of both. For errors, the front of the response message is considered when addition of the user is not perfect. However, for this simple system no errorcode is required. Server just returns the error message or response as a string to the client.

4. Label Semantics

- Identifying both users and rooms involves sending and receiving labels. All label rules are the same, and MUST be validated as follows:
- Must consist entirely of readable utf character values.
- Must be at least 1 character, and at most in 100k input size.

5. Client Messages

On the client side, the user will send particular commands with the arguments to the server.

5.1 Welcome client
- Usage : person will enter his username
- Response - welcome user

5.2 List all rooms
- Usage : person will enter command “liro”.
- Response - list of all rooms

5.3 List all room members
- Usage : person will enter command “lime roomname”.
- Response - list of all members in the room.

5.4 Create a room
- Usage : person will enter command “room roomname”.
- Response - room with given name will be created.

5.5 Join a room
- Usage : person will enter command “join roomname”.
- Response - client will join the room with a given name.

5.6 Leave a room
- Usage : person will enter command “leve roomname”.
- Response - client will leave the given name room.

5.7 Send message in a room
- Usage : person will enter command “send roomname message”.
- Response - client will receive the confirmation message that message to given roomname is delivered.

5.8 Get help
- Usage : person will enter command “help”.
- Response - client will receive the text describing all the available commands.

5.9 Exit the running client
- Usage : person will enter command “exit”
- Response - person will stop the execution of the current client process and he will be prompted with the new terminal command instance.

5.10 Error responses
- Some examples of the error response

```
INVALID_ROOM_NAME     = "invalid room name"
ROOM_DOES_NOT_EXIST   = "room does not exist"
NOT_MEMBER            = "you are not member"
INVALID_MESSAGE_FORMAT = "invalid message format"
```

6. Server Messages

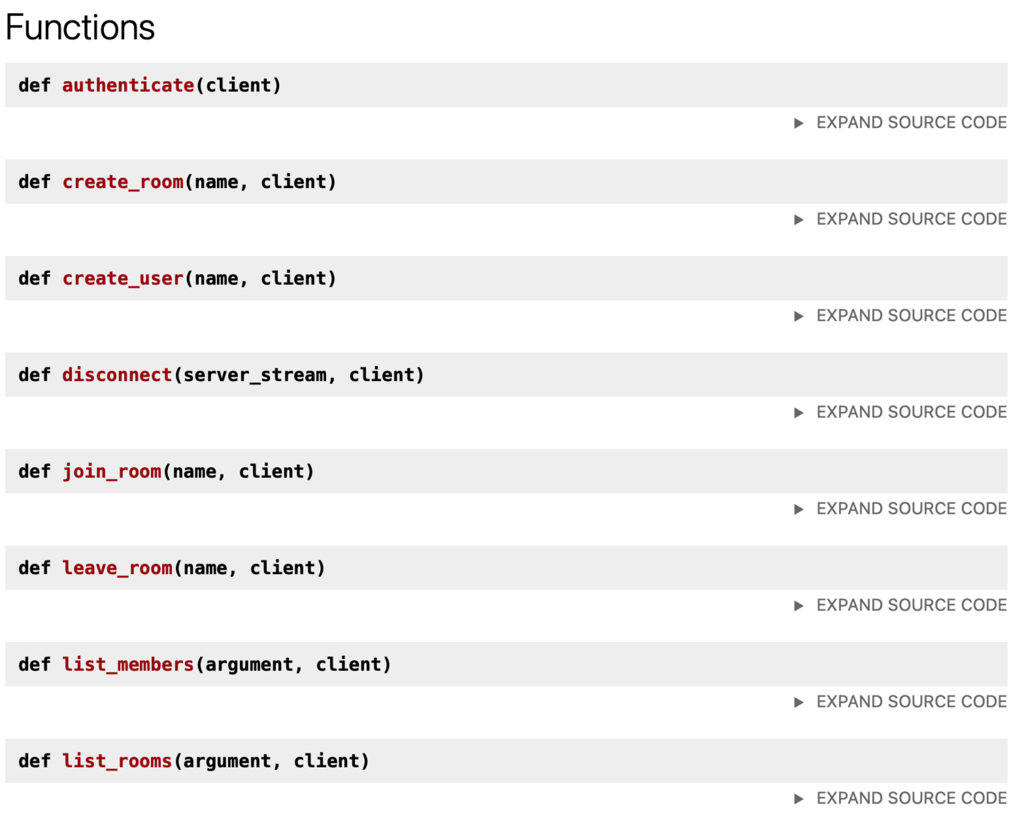
On the server, there are responses as per the given client command or error message. For example, if the username is smaller than two characters then output will show “invalid username”.If you write exit in the command line terminal of the server then the server program will stop execution.

For example, all the server messages are here.

```
SERVER_STARTED       = "server started: ctl + c to exit or exit to exit"
SERVER_STOPPED       = "server stopped"
NEW_CLIENT           = "new client"
WELCOME_CLIENT       = "welcome user \n"
CLIENT_INPUT         = "client input received"
CLIENT_INVALID       = "client invalid"
CLIENT_DISCONNECT    = "client disconnected"
CLIENT_DISCONNECT_ERR = "client disconnect error"
CLIENT_ALREADY_IN    = "you are already in the system"
CLIENT_EXISTS        = "name exists"
INPUT_INVALID        = "input invalid"
UNKNOWN_COMMAND      = "command invalid Type - HELP"
ROOMS_AVAILABLE_TITLE = "Available rooms \n"
NO_ROOMS_TITLE       = "Sorry, no rooms "
INVALID_ROOM_NAME    = "invalid room name"
ROOM_DOES_NOT_EXIST  = "room does not exist"
ROOM_MEMBERS         = "room members \n"
ROOM_ADDED           = "new room added"
ALREADY_MEMBER       = "you are already member"
NEW_MEMBER_JOINED    = "new member joined in room"
MEMBER_LEFT          = "one member left the room"
YOU_LEFT_ROOM        = "you left the room"
MEMBERSHIP_GRANTED   = "Membership granted to the room"
NOT_MEMBER           = "you are not member"
INVALID_MESSAGE_FORMAT = "invalid message format"
EXIT_SUCCESSFUL      = ""\n exit successful"
```

7. Manager Module

This module manages management of different command executions



Functions

```
def authenticate(client)
```

▶ EXPAND SOURCE CODE

```
def create_room(name, client)
```

▶ EXPAND SOURCE CODE

```
def create_user(name, client)
```

▶ EXPAND SOURCE CODE

```
def disconnect(server_stream, client)
```

▶ EXPAND SOURCE CODE

```
def join_room(name, client)
```

▶ EXPAND SOURCE CODE

```
def leave_room(name, client)
```

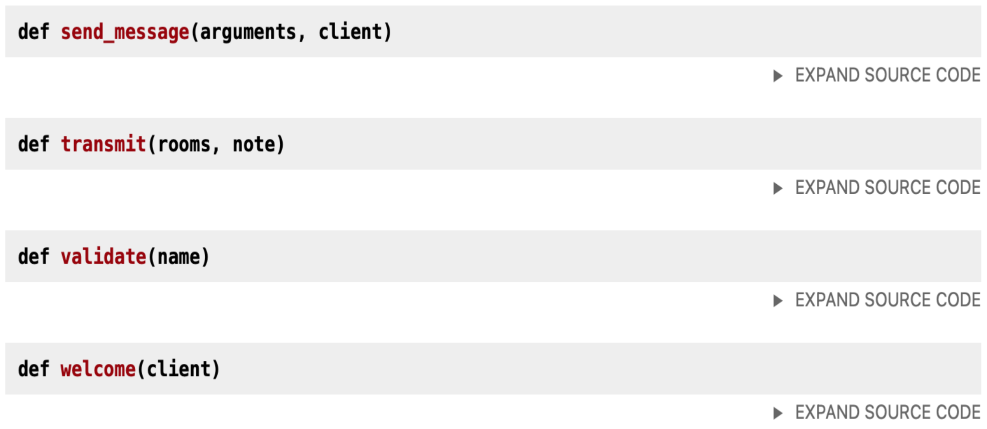
▶ EXPAND SOURCE CODE

```
def list_members(argument, client)
```

▶ EXPAND SOURCE CODE

```
def list_rooms(argument, client)
```

▶ EXPAND SOURCE CODE



```
def send_message(arguments, client)
```

▶ EXPAND SOURCE CODE

```
def transmit(rooms, note)
```

▶ EXPAND SOURCE CODE

```
def validate(name)
```

▶ EXPAND SOURCE CODE

```
def welcome(client)
```

▶ EXPAND SOURCE CODE

7. Conclusion & Future Work

This specification provides a generic message passing framework for multiple clients to communicate with each other via a central forwarding server. Moreover, in future, private messaging, secure messaging, file transfer and implementation on cloud server can be completed.

8. Security Considerations

Messages sent using this system have no protection against inspection, tampering or outright forgery. The server does not see all messages that are sent through the use of this service.

9. IANA Considerations

None

9.1 Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10 Acknowledgements

This document was prepared using google doc.