RFC IRC CHATBOX SERVER AND CLIENT

Cesia Bulnes
5291722

1. Introduction

The Chatbox server and client assignment is one needed to conform to the Internet Relay Chat: Client Protocol creating the abilities for several contributions of transferring data between clients, with the receiver having to abide specific instructions when handling data. This specific IRC protocol is used for text-based data exchange, with the simplest client having a socket program that can connect to the server. In order to follow the IRC Protocol one needed to abide by the specific rules for each command assigned. This led to parsing of arguments, and hashing many of them by creating a complex network of stored information. This program was coded using python, therefore following the python documentation rigidly helped me improve my coding skills with it. Looking up IRC Commands was also necessary in order for the chatbox to work proficiently. The final product of this project reveals a chatbox in which different clients can connect to a server and store information and exchange messages/data. I would say that the biggest flaw of the project was the inability for users to send messages in private windows. Other than that, users would be able to communicate with one another and obtain information about other users and/or channels within this chat. The most important websites used to abide to the instructions were the following:
https://docs.python.org/2/howto/index.html
https://tools.ietf.org/html/rfc2812
https://en.wikipedia.org/wiki/List_of_Internet_Relay_Chat_commands

2. Problem Statement

The main problem to be solved was to create a working server that would manage and store data given through the client, and then to be able to execute methods or commands, that would retrieve information and/or store it. These commands were sometimes difficult to implement because they had to abide by a certain protocol or standard.

3. Methodology

Because I started this program earlier than most, I was too deep into it to use User.py or Channel.py. I basically created many hashmaps that led to my db class.

```python
class DB:

    def __init__(self, db_path):
        self.USERS_FILE = db_path + "/users.txt"
        self.BANNED_USERS_FILE = db_path + "/banusers.txt"
        self.CHANNELS_FILE = db_path + "/channels.txt"
        self.BANNER_FILE = db_path + "/banner.txt"

        # username -> {username: str, password: str, level: str, banned: bool}
        self.users = {}
        # str[]
        self.banned_users = []
        # name -> {name: str, description: str, password: str, channelops: str[]}
        self.channels = {}
        # str
        self.banner = None

        self.refresh_from_files()
```

from this DB class, I was able to store information to db files such as channels.txt.These hashmaps also stored information such as the password, username, channelops, etc. I also included information and more hashmaps into hashmaps in the Server class. These hashmaps generally did not require to be written into the text files. Hence, why I kept it out of my DB class. But mainly the information was stored in hashmaps throughout my program, and I created

a DB class from scratch in order for this information to be stored in the text files as asked.

```
                allow_reuse_address=True):
        self.host = host
        self.port = port
        self.address = (self.host, self.port)
        self.clients = {}
        self.client_thread_list = []

        # key: name, value : address
        self.client_ips = {}
        # key: name, value : username
        self.client_usernames = {}
        # clients who are away
        self.clients_away = {}

        # key: nickname, value: name
        self.clients_nicknames = {}
        # key : name, value : password
        self.clients_passwords = {}

        # users, banned users, channels and banner db
        self.db = db
        # key: channel_name, value: name[]
        self.channel_users = {}
        # key : mode, value: name[]
        self.mode_of_users = {}
        # key : channel name  , value : topic
        self.channel_topic = {}

        self.online_users = []

        # clients who will be silenced
        # key: username of person blocking , value: name[] of people being blocked
        self.ignore_list = {}
        try:
```

From there, each command was coded based on the IRC, and the parameters revolved on what was needed for each command.

As for the client, the main part in my opinion was creating argument parsing for each argument passed in the command line.

```
argument_parser = argparse.ArgumentParser("IRC Chat Client")
argument_parser.add_argument(
    "-hostname",
    help="Hostname of the IRC Chat Server which the client should connect to",
    type=str
)
argument_parser.add_argument(
    "-u",
    help="Username of the IRC Chat Server which the client should claim itself as",
    type=str,
    required=True
)
argument_parser.add_argument(
    "-p",
    help="Port of the IRC Chat Server which the client should connect to",
    type=int
)
argument_parser.add_argument(
    "-c",
    help="Path of the Configuration file",
    type=str,
    required=True
)
argument_parser.add_argument(
    "-t",
    help="Test File",
    type=str
)
argument_parser.add_argument(
    "-L",
    help="Log file name, for log messages",
    type=str
)
```

I also want to point out that I was able to log my information throughout the making of this program.

4. Results

I will provide some results by running some commands through a test_file.txt. This simply means that the commands will all be executed at the same time and produce the outputs up on the client GUI.

The following is the Test_file, followed by the results on the GUI.

ChatServer.py | channels.txt | ChatClient.py | Main.py | test_file.txt

[PyGuiChat2] ~/Desktop/netCentric/PyG

```
/pass pass
/user cesia 2 * cesia
/nick cesi
/mode cesi + w
/wallops message
/away message1
/ison cesi
/rules
/version
/join channel1
```

ChatApp

```
> Welcome to our chat app!!! What is your name?
> Welcome cesia, type /help for a list of helpful commands.

Successfully set new password!

You have successfully been set as a user


Successfully set new nickname!:

cesiyour mode has been added

Message to Wallops : message

Your message has been sent to users users with modes operators and wallops

> Away message was set


ISON NICKNAMES currently on IRC:cesia


                        > The rules of the chat are as follows:
                        Be nice to my grade


The version of this chat is version 1.0


Added you to channel channel1
```

5. Analysis

My experience with this program has been pretty frustrating simply because I started from scratch, by even implementing my own DB class. Though I was unable to create new windows

or buffer information, I feel like my knowledge with python, servers, and clients in general has increased tremendously. The program seemed never ending with so many commands but the satisfaction of coding all of them was a feeling of accomplishment that I haven't felt before in my undergraduate experience. I definitely hadn't worked with hashmaps as closely, so a lot of the time I was having tiny bugs because of type errors. I definitely did enjoy this project as self-taught knowledge is the one that sticks with people in general.

6. References

"Python HOWTOs¶." *Python HOWTOs - Python 2.7.14 Documentation*, docs.python.org/2/howto/index.html.

"Internet Relay Chat: Client Protocol." IETF Tools, tools.ietf.org/html/rfc2812.