Chat_Server.py

Author: Nick Francisci

Tester: Deniz Celik

Overview:

The chat server that I created for my group serves the primary purpose of relaying messages between two connected clients. In order to do this, the server supports user commands, parsed from received messages, including a log-in command that aliases each user's IP with a username. In addition to a log-in/log-out system, the server supports admin functionality that is separated from user functionality via a simple password. A full glossary of user and admin functions is included in the latter sections of this document.

Process of Creating the Server:

Note: What follows is a brief accounting of the major steps towards implementing the primary features of the server in its current incarnation. It is not exhaustive and omits many of the available functions that are implemented and also omits stretches of time spent debugging and the time spent writing small bits of code that protect the server from being crashed by users or executing commands in undesirable ways.

Basic Functionality

The key functionality of a chat server is its ability to relay messages between two or more people. In order to implement this core functionality, I started with the receiver script that was given to the class. I then ported over the key parts of the

sender script that was given to the class and functionalized it for use by the receiver script. I then added a simple list which logged the IPs of all incoming messages that were not from an IP already on the list. Then, I added a function to relay a message to all IPs on the list with a sender address marked by the IP of the sender that the server had received it from. Finally, I set up the server to call the relay message function for every incoming message. With these steps complete, the basic functionality of the chat server was implemented and the server was operational.

Introduction of Command System and Users

The first major addition to the functionality of the server started with the goal of aliasing the IP of a sender to a username. In order to do this, I had to allow the user to choose an alias and have the server recognize the submitted data and store it for future use. Recognizing that this was the beginnings of a basic user command system, I implemented \ as an escape character by telling the server to refer all incoming messages beginning with a \ to a newly created user-commands class. The user-commands class looked at the first word after a \ and determined this to be the command, with the subsequent word (if one existed) as an argument to that command. This command and argument were then parsed to the appropriate function.

As part of the log-in/out functionality I also moved away from a list of IPs to a dictionary containing IP key values and values of class User which were built to contain an alias and additional user information for use in future functions. Using the Users dictionary and the user-command system together, I rewrote some of the server relay logic to check if a client was logged in by their IP and, if not, requested that they create an alias before the server would relay their messages. Then, by checking the IPs of the sender, messages were relayed under the name of the sender IP's registered alias.

Addition of Admin System

In the running of a chat server, some amount of administration is sometimes desirable: it may be that the server log needs to be viewed, that a user needs to be banned for offensive behavior, or, in general, that some interaction directly with the server is necessary. The server code did not support an interface for taking in commands during runtime, and multithreading a command terminal of this nature seemed unnecessary for the functions needed. Therefore, I decided to use the existing user-command system to submit admin commands, so long as the user was validated as an admin. To this end, I added an admin boolean to each user, a string password to the server, and a function to match an admin command with a password argument against the string password to authenticate a user as an admin. Then I added some control logic to check if a submitted user's command was both an admin function and that they were an admin to determine whether functions from the admin commands class were executed.

Future Features and Fixes

In addition to the features already implemented, there is a long list of improvements to the server that remain to be made. Some of the major ideas that come to mind are:

- Password hashing (so that admin passwords are not sent in plaintext over the network
- Metering posts per unit time for a user to prevent spamming or brute-forcing of the admin password
- Metering time inactive for users so that they are automatically logged out and messages are no longer relayed to them.
- An invisibility attribute for each user that allows each user to toggle whether or not their alias is returned when another user asks for the list of users in the chat-room
- Periodically saving server logs to file

Command Appendix: Users

\admin [pw]	If the password is correct, toggles the user's admin status
\connect [alias]	Adds the user to the chat-room under the given alias
\disconnect	Removes the user from the chat-room
\help	Displays a list of available commands
\setName	Resets user alias
\users	Displays a list of (non-invisible) users in the chatroom

Command Appendix: Admin

\banUser [alias]	Bans the user with the specified alias until ban is lifted
\clearServerLog	Deletes server log
\dispBannedIPs	Displays the IPs of all banned users
\liftBan [IP]	Lifts ban on IP specified and notifies them
\printServerLog	Prints as many of the most recent entries in the server log as the buffer size allows to be sent in a message