Henry Stiefel

Trent Maas

Programming Assignment 2 README

This is my Client-Server Message Board Project! Both the client and server programs are written in Python – they can be ran just as you would run any other Python script.

**Server**

The server program simply needs to be run, and by default it will start listening for client connections on localhost:9999. This data is hardcoded into the script, but could be easily changed if you wished to use the server program on a live network with port forwarding. The server has been tested to work with 5 groups (hardcoded) and up to 3 local client connections.

**Client**

The client program also needs to be run, and then the user must use the Python terminal to send commands and communicate with the server. The client should start by typing “CONNECT {ip} {port}”, or in our test case, “CONNECT 127.0.0.1 9999”. This will connect if a server is listening on that ip and port. Next, the user will be prompted to enter a username. They must do so by typing “USERNAME {your username here}”, for example “USERNAME bob”. They will then receive a welcome message from the server, as well as the list of other users, previous messages (if there are any), and a list of groups to join. Once connected, the user is free to use any of the commands to interact with the server and other clients. **There is a detailed list of all commands and their uses/arguments, as well as what they do below**. If the user wishes to leave the server at any time, they may type “LEAVE” to close their connection with the server, or “EXIT” to stop the running of the program.

Below is a list of commands and how to properly use them. The program should catch any uses of incorrect parameters and notify the user accordingly.

Commands (case sensitive):

* COMMANDS – this will display a list of all commands the client can use
  + Returns plaintext list of commands and their parameters
* CONNECT [ip] [port] – this will connect to server on the given ip/port, if there is any
  + Sample use: “CONNECT 127.0.0.1 9999”
* USERNAME – should only be used once, when prompted by server to join, but allows the client to tell the server their username (one word limit only)
  + Sample use: “USERNAME bob” (when prompted by server)
* POST [subject] [content] – posts a message to the public/general message board. The subject is 1 word only, and anything following the subject will be considered part of the content. Other users will be notified that a new message has been posted with the format: “Message ID, Sender, Post Date, Subject”
  + Sample use: “POST Weather So, how do the rest of you like the weather today?”
* USERS – displays a list of all other usernames connected to the server
  + Returns: plaintext list of the usernames of other clients
* LEAVE – leaves the server the client is currently connected to
* MESSAGE [message id] – retrieves the message with the given id, as long as the id is valid. This command will return only the subject, date, and content of the message.
  + Sample use: “MESSAGE 1”
  + Returns: “Subject: Weather Date: 10:24 Content: So, how do the rest of you like the weather today?”
* EXIT – exits the current running client program completely
* GROUPS – displays a list of the 5 groups on the server. These are hardcoded into the server code as group 0, 1, 2, 3, and 4.
  + Returns: plaintext list of group IDs 0 through 4
* GROUPJOIN [group id] – joins the group with the given id, if the id is valid
  + Sample use: “GROUPJOIN 3”
* GROUPPOST [group id] [subject] [content] – behaves similarly to the regular POST command. The user must be a member of the group to post in that group, and only other group members will be able to access this new post. The subject is 1 word only. The content is considered anything from the fourth word and on in the command.
  + Sample use: “GROUPPOST 3 Grades I am worried about my grade in this class, how about you guys?”
* GROUPUSERS [group id] – displays a list of users that are members of the given group
* GROUPLEAVE [group id] – leaves a group if the user is a member of the group with the given id
* GROUPMESSAGE [group id] [message id] – retrieves a message in the format “Subject, Date, Content”. Note, message ids start at 0 for each group (including the public one) and increment from there, so for example both group 0 and group 1 could have a message with id=1 (but the messages will be unique per each group).

**Major Issues**

The biggest issue that we came across in development was how to allow the client to both listen for messages from the server and also listen to the input stream through the client terminal. Apparently, there are issues that are easily handled by Linux and Mac, but cause problems on Windows machines (which I developed on). I ended up using two separate threads for the client program, one to listen and one to take input. Another major issues was how the server was able to detect if a client left the server. To get around this, I have the server send out a ping to each client every so often. If the server’s ping gets no response or is not sent successfully, it assumes the client has timed out or terminated the program, and so it removes it from the server (and notifies other users that it has been removed).