MP1 Documentation

1. Client – Server Chat Application

This application has 2 programs, the server and the client. The goal is to create an application that would allow the connection of multiple users from different computers and be able to send messages to the group as a whole. The application must also be able to handle a few commands coming from the users and perform them accordingly, meaning that there must be a way to differentiate commands from messages.

The entire application is made using Python 2.7 with TCP socket programming.

1. Server Program

The program where all the connections and message exchange is handled. It has two defined functions, broadcast() and command(), and a main program.

Using TCP, a server socket is opened and waits for up to 10 (as a default but can be changed) client connections. A list for all connections and two dictionaries for the username and user data are established at initialization. The server has a dynamic port which the admin of the server can input at initialization also.

To handle the message exchange between multiple clients, a loop is created which reads through the connection list at each cycle. First, it checks if there is a new connection using the socket.accept() function and then stores the socket to the connection list to be read through at the following loop cycles. It also stores the IP address and the port of the new connection to a dictionary having the socket as the key or identifier. Later on, the identifier will be changed to the client’s username. The username is stored in another dictionary with the same socket as key. As a default, the connection’s IP address and port serves as the initial username in the format (‘0.0.0.0’, 00000). As the loop iterates through each socket connection, it checks if that connection has sent a new message. If so, the server sends the message to all the other connecting sockets using the defined broadcast() function.

broadcast() function – this function is another loop which cycles through each socket in the connection list. It checks whether the iterated socket is the server socket or the socket which sent the message. If not, it will then send the message accordingly. If it is unable to send the message through the socket, it assumes that the socket has been unexpectedly closed and thus removes the respective socket from the lists and alerts all other clients.

If the server detects that the message is a command, it will call on the defined command() function. This handles how to respond to each individual commands.

command() function – this function breaks the message string into a list of words. Because of the format of the agreed command protocol (which will be explained later on), the first word on the list is recognized as the command identifier. The server has a list of commands to which it will compare the client message to, if it is within the existing commands, the server will respond accordingly. Right now, this server accepts only the quit, name, whois, time, and help command. This function also handles errors such as unrecognized commands and the use of similar usernames.

To close the server, the only available method is through KeyboardInterrupt or ctl+C which will then close the server sockets and signal the clients to close as well.

1. Client Program

The program users will use to connect to the server. Its purpose is mainly to send and receive data to and from the server. At execution, the user will be prompted for the IP of the host server and also the corresponding port. This will create a client socket to that server.

For this program to work, it must be able to check for incoming messages from the server while simultaneously checking for user input from the system and then send it to the server. In order to do this, a list consisting of sys.stdin (system user input) and the client socket is created at initialization. A loop is created that iterates through these two at every cycle. In the loop, it checks the client socket for received messages from the server then prints it out. It also handles whether the client socket is unexpectedly closed. Else, it checks for user input. If there is user input, it will send it to the server.

1. Client - Server Section Protocol

The following is the protocol followed in the application as is decided by the Friday section.

Commands vs. Message – in order for the server to recognize that the sent string is a command, it must be preceded by a ‘/’ character. Therefore, the format is /<command> (without the braces). If the string is not preceded by the character, it will be handled as a message.

The following are the commands agreed upon and as specified in the MP:

1. /name <name> #this command assigns the string inside the <name> as that clients username
2. /quit #exits the program
3. /whois <name> #returns the IP and Port used by that username to the requester
4. /time #returns the server time

Message format – Since the usernames are all stored and kept track of in the server, the server sends all messages in the format ‘Username: message’ such that the client will only have to print the whole message sent by the server. It was also agreed upon that all messages sent by the server must not have a trailing new line (\n).

1. Peer-to-Peer Chat Application