Project 1: Chat Application

Project Description

A chat program is an application that enables communication between multiple chat clients. There are two typical paradigms to enable such communications: client-server and peer-to-peer communication. For the client-server paradigm, it is the responsibility of the server to relay the messages that are exchanged between clients.

For this project, you need to develop a UDP-based chat application, following the client-server communication paradigm. This application will use sockets for communications. Each functionality of this chat application is described below.

Sockets Communication

Write a simple client-server broadcast chat application. The server listens at a specified UDP port, and waits for `GREETING` messages from remote clients. Once a greeting has been received the client may send a `MESSAGE` command to the server, which will forward the contents in an `INCOMING` message to every remote point that has previously sent a `GREETING` (including this sending client).

The types and format of messages to be exchanged in the application are:

1. `GREETING`: Greets the server. Client to server only. The server should register the client who sent this message as active.
2. `MESSAGE`: Sends some text to the server for further distribution. Contains the text of the message to send. Client to server only.
3. `INCOMING`: The server has received some text, and is passing it along with the sender's ip and port. Contains the IP address and port of sender as well as the text of the message. Server to client only.
4. Any other message (e.g. ICMP errors) must be ignored.

A sample run of your application must work as follows:

- `server$ python ChatServer 9090` runs the server on port 9090
  `Server Initialized...` Server is left running
- `user1$ python ChatClient server-ip 9090` runs the client and GREETs the server.

`]` Prompt message from user
`] Hello World!` Sends a message
`<From w.x.y.z:aa>: Hello World!` Every client sees this, where w.x.y.z:aa is the sender's ip address and port.
`]`

Note the following:

- The messages do not need to be authenticated, confirmed nor encrypted.
- There are no usernames whatsoever.
- The server sends the message to everyone who has GREETED him. There is no mechanism to make the server "forget" or logout clients.
- There is no limit to the number of clients the server supports.
- The client needs to accept messages from both standard input as well as sockets. Consider using I/O multiplexing functions, such as `select` and `epoll`, to handle this.
- You can use any programming language for this assignment.

Useful Links

Programming in C

If you choose to write your programs in C, you'll need to master the C system calls needed for socket programming. These include *socket(), bind(), listen(), accept(), connect(),* and *close().* For a quick text-only tutorial of socket programming specifically under Linux, see http://www.lowtek.com/sockets/. Here's another nice tutorial: http://beej.us/guide/bgnet/

Programming in Python

The 8th edition of the text (and what we've discussed in class) uses sockets in Python. A Python socket tutorial is http://docs.python.org/howto/sockets.html

Programming in Java

A Java socket tutorial is https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

REMINDERS

- Submit your code to Canvas in a zip file
- If your code does not **compile** or could not **run**, you will not get credits.
- Document your code (by inserting comments in your code)
- DUE: 11:59 pm, Monday, Feb 20th.