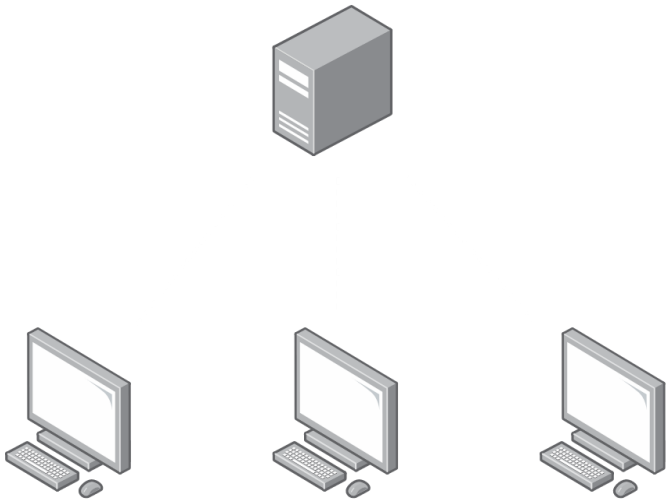
DETAIL OF WHOLE PROGRAM FOR INITO.

INTRODUCTION:
Python is a great programming language for computer networking. It allows us to create solid applications very fast and easily. In this tutorial, we are going to implement a fully-functioning Client -Server. We will have one server that hosts the text and multiple clients that connect to it and communicate with each other. At the end, you can also add custom features like chat rooms, commands, roles etc., if you want to.

Client Server Architecture:

For our application, we will use the client-server architecture. This means that we will have multiple clients (the users) and one central server that hosts everything and provides the data for these clients.



Therefore, we will need to write two Python scripts. One will be for starting the server and one will be for the client. We will have to run the server first, so that there is a chat, which the clients can connect to. The clients themselves, are not going to directly communicate to each other but via the central server.

Implementation of Server

Now let's start by implementing the server first. For this we will need to import two libraries, namely *socket* and *threading*. The first one will be used for the network connection and the second one is necessary for performing various tasks at the same time.

The next task is to define our connection data and to initialize our socket. We will need an IP-address for the host and a free port number for our server. In this example, we will use the localhost address (127.0.0.1) and the port 11826. The port is actually irrelevant but you have to make sure that the port you are using is free and not reserved. If you are running this script on an actual server, specify the IP-address of the server as the host. Check out this list of reserved port numbers for more information.

When we define our socket, we need to pass two parameters. These define the type of socket we want to use. The first one (*AF_INET*) indicates that we are using an internet socket rather than an unix socket. The second parameter stands for the protocol we want to use.
After defining the socket, we bind it to our host and the specified port by passing a tuple that contains both values. We then put our server into listening mode, so that it waits for clients to connect. At the end we create two empty lists, which we will use to store the connected clients and their nicknames later on.

Here we define a little function that is going to help us broadcast text and makes the code more readable. What it does is just send a text to each client that is connected and therefore in the clients list. We will use this method in the other methods.

Now we will start with the implementation of the first major function. This function will be responsible for handling text from the clients.

As you can see, this function is running in a while-loop. It won't stop unless there is an exception because of something that went wrong. The function accepts a client as a parameter. Everytime a client connects to our server we run this function for it and it starts an endless loop.

What it then does is receives the text from the client (if he sends any) and broadcasts it to all connected clients. So when one client sends a text, everyone else can see this text. Now if for some reason there is an error with the connection to this client, we remove it and its ni, close the connection and broadcast that this client has left the chat. After that we break the loop and this thread comes to an end. Quite simple. We are almost done with the server but we need one final function

When we are ready to run our server, we will execute this *receive* function. It also starts an endless while-loop which constantly accepts new connections from clients. Once a client is connected it sends the string *'Hii* to it, which will tell the client that its nickname is requested. After that it waits for a response (which hopefully contains the nickname) and appends the client with the respective nickname to the lists. After that, we print and broadcast this information. Finally, we start a new thread that runs the previously implemented handling function for this particular client. Now we can just run this function and our server is done.

Notice that we are always encoding and decoding the messages here. The reason for this is that we can only send bytes and not strings. So we always need to encode messages (for example using ASCII), when we send them and decode them, when we receive them.

A server is pretty useless without clients that connect to it. So now we are going to implement our client. For this, we will again need to import the same libraries. Notice that this is now a second separate script.

The first steps of the client are to choose a text and to connect to our server. We will need to know the exact address and the port at which our server is running.

As you can see, we are using a different function here. Instead of binding the data and listening, we are connecting to an existing server.

Now, a client needs to have two threads that are running at the same time. The first one will constantly receive data from the server and the second one will send our own text to the server. So we will need two functions here. Let's start with the receiving par

Again we have an endless while-loop here. It constantly tries to receive messages and to print them onto the screen. If the text is 'Hii" however, it doesn't print it but it sends it to the server. In case there is some error, we close the connection and break the loop. Now we just need a function for sending text and we are almost done.

The writing function is quite a short one. It also runs in an endless loop which is always waiting for an input from the user. Once it gets some, it combines it with the nickname and sends it to the server. That's it. The last thing we need to do is to start two threads that run these two functions.

And now we are done. We have a fully-functioning server and working clients that can connect to it and communicate with each other.

Thanks & Regards
Sachin Giri