

# Chat Session Server

## Server Program

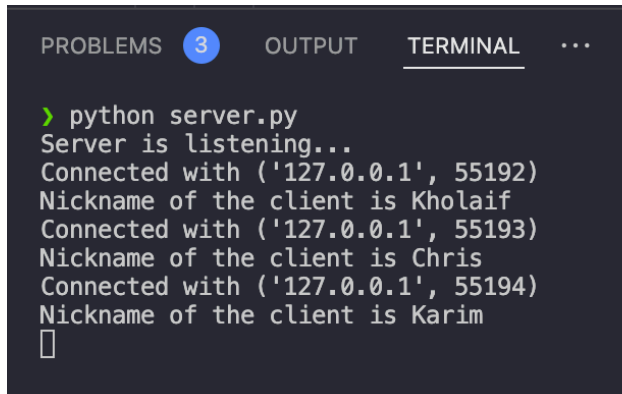
The server Python program is the server side of the pair of applications, which essentially hosts the chat room. First, we configure the server with a host and port number, creating a socket with specified parameters for IPv4 and TCP. We then bind the server socket and begin listening for incoming connections. We declare two lists to store client sockets and the respective nicknames of clients. Two functions are then declared, 'receive()' and 'handle()'. The first function uses an infinite loop to continuously accept incoming connections from clients, seeing a message to the client requesting a nickname and storing it in the respective list. The second function runs in a loop, receiving messages from connected clients. If a client disconnects, the client is removed from the respective list and connections are closed. Threads are used for the server to handle multiple clients concurrently.

## Client Program

The client program first allows the user to choose a nickname to identify the client in the chat. The client then initializes a socket with parameters for IPv4 and TCP, then connecting to the server using the server's address and port number. The 'receive()' function runs in an infinite loop, continuously receiving messages from the server. If the received message is "NICK", then the server is requesting the client's nickname, so the client sends the chosen nickname to the server. If the received message is not "NICK", it's a regular chat message that is printed to the client's console. To send messages to the server, the write() function also runs in an infinite loop, continuously prompting the user to input messages. The user's input and nickname are formatted into a message string that is sent to the server. Threading is used to handle receiving and sending messages concurrently, with a thread handling receive and another thread handling write.

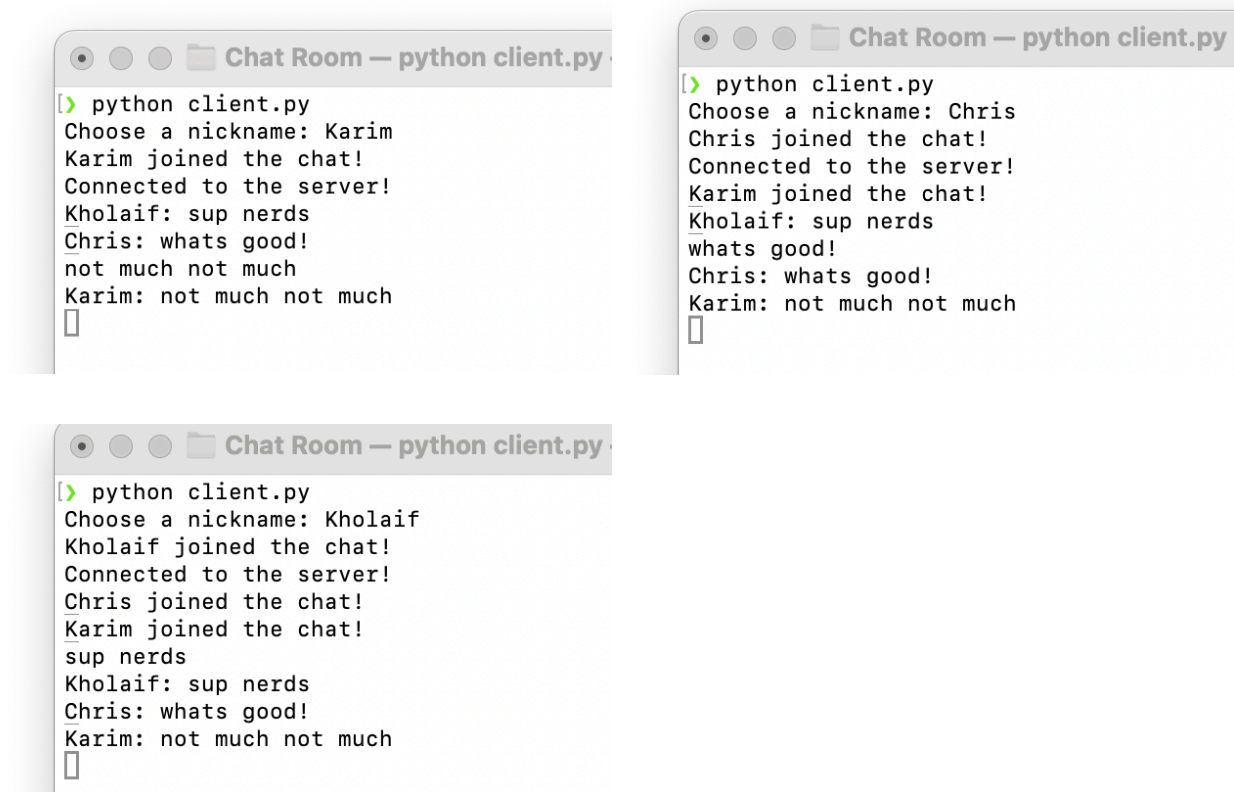
## Program Operation

Below are screenshots of the server being initiated in the terminal with three clients connecting to it:



```
PROBLEMS 3 OUTPUT TERMINAL ...  
  
> python server.py  
Server is listening...  
Connected with ('127.0.0.1', 55192)  
Nickname of the client is Kholaiif  
Connected with ('127.0.0.1', 55193)  
Nickname of the client is Chris  
Connected with ('127.0.0.1', 55194)  
Nickname of the client is Karim  
█
```

The terminals for the individual users are below:

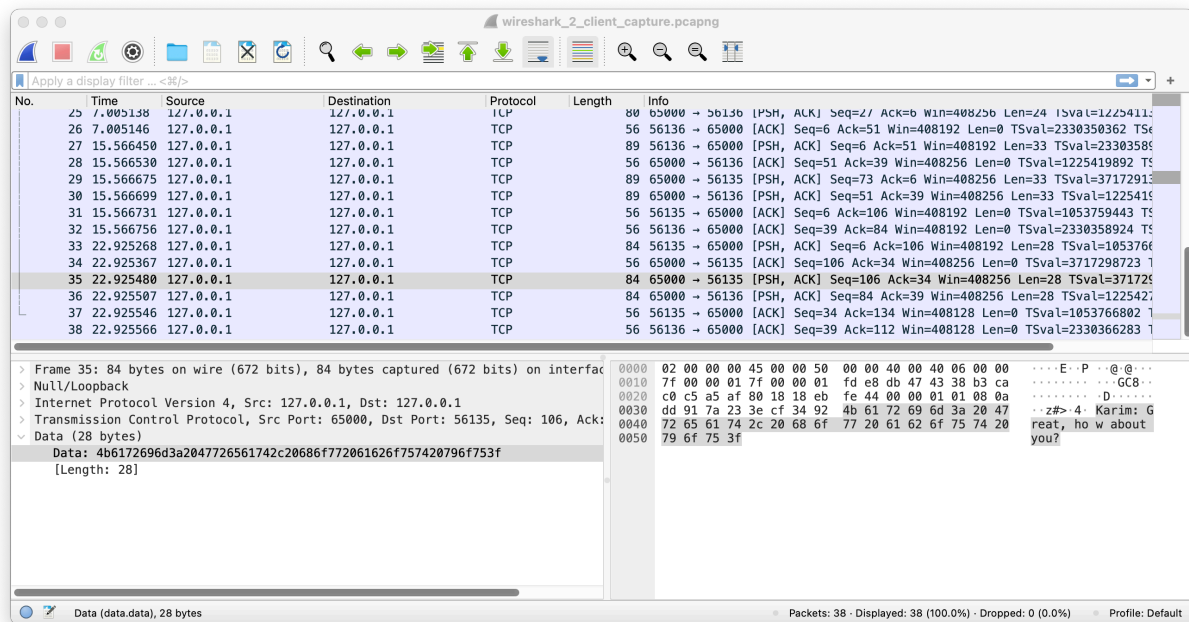


```
Chat Room — python client.py  
[> python client.py  
Choose a nickname: Karim  
Karim joined the chat!  
Connected to the server!  
Kholaiif: sup nerds  
Chris: whats good!  
not much not much  
Karim: not much not much  
█
```

```
Chat Room — python client.py  
[> python client.py  
Choose a nickname: Chris  
Chris joined the chat!  
Connected to the server!  
Karim joined the chat!  
Kholaiif: sup nerds  
whats good!  
Chris: whats good!  
Karim: not much not much  
█
```

```
Chat Room — python client.py  
[> python client.py  
Choose a nickname: Kholaiif  
Kholaiif joined the chat!  
Connected to the server!  
Chris joined the chat!  
Karim joined the chat!  
sup nerds  
Kholaiif: sup nerds  
Chris: whats good!  
Karim: not much not much  
█
```

To showcase the packets being transferred between clients and server, below is a screenshot of TCP packets captured by Wireshark. The capture file for 2 clients (Karim and Chris) and the server is also shared in the zip file for reference. In the screenshot below, pay attention to the actual message in the data portion of the packet, showing the message from Karim responding with “Great, how about you?”:



## Analysis of Program Functionality

There are several comparisons that we can draw between this chat application and a typical text message application on a phone. Similarly to a smartphone messaging app, the client-server architecture allows users to connect and communicate over a network with each other. The difference here is that connection is explicitly established between a client and server. Messages here are sent from the client to the server, which then relays the messages to other clients. Smartphones may use a centralized server or peer-to-peer connections for exchanging messages, depending on the type of program. Smartphones tend to also have very well-designed interfaces for chat programs, whereas this entire program relies on terminals and command-line interfaces. Both programs provide real-time communication, although the responsiveness will depend on latency. Typically, smartphone applications with better infrastructure will not be as affected as this simplistic program.

Some of the advantages of this client/server Python chatroom application are simplicity, customization, privacy, and more. The chat application is incredibly lightweight, making it easy to deploy and use anywhere. It's suitable for scenarios where a lightweight messaging solution is best, capable of running using minimal device resources. The program is also very customizable, since users have far

more control over the application's behavior, as they can modify the code to extend or customize functionality according to individual needs. Anyone who knows Python can add their own features to the program. Privacy and security are also an advantage of this program, as the chat application is self-hosted with no reliance on third-party servers or authentication.

Some limitations of this program are limited user experience, dependency on connectivity, and scalability. The command-line interface of the application doesn't offer the same user experience as graphical messaging applications that exist. Additional features need to be manually added, so features like media sharing, emojis, and more aren't available by default. This program requires internet access to send messages rather than cellular access, requiring that users have an internet connection. The scalability of this program is also quite poor, as the application may face challenges with a large number of users, with degradation of performance. In a small environment, this application is great.

### Usefulness and Applications

This chat application serves as a valuable tool for students like me to learn and understand network programming concepts, socket communication, and multithreading. Simplicity and control are valued over features that are hip and cool, allowing students to focus on learning the fundamentals rather than fancy interfaces and APIs. This application can also be built upon, with a simple interface at the beginning allowing students to add any features they can think of on top. Students can use this program to experiment with different networking protocols, security measures, and optimization techniques.