Client/Server Reliable chat Application

Project Report

By

Ali A. Zamin

Abstract

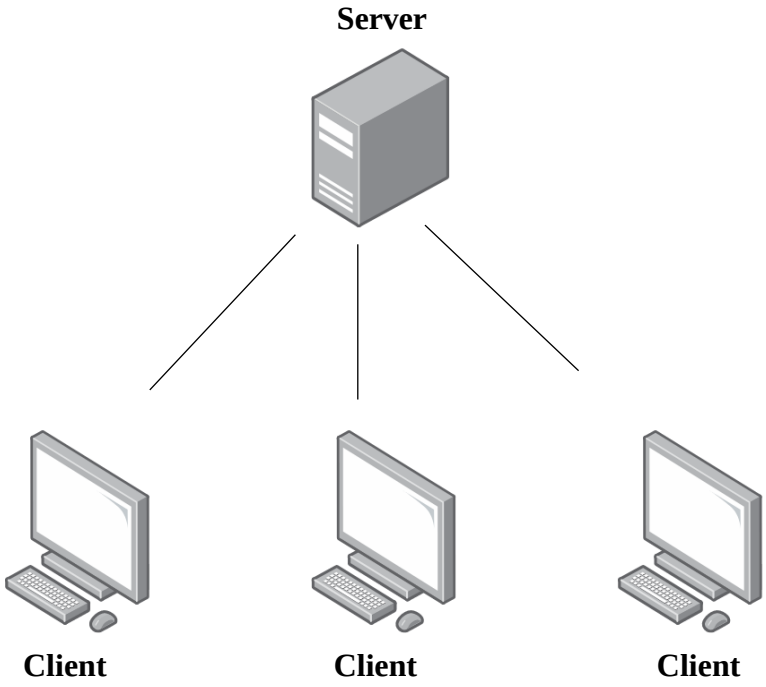
In this project, we will build a Multiple Client/Server chat application using socket programming and Multithreading, that implements client-server communication. The application will support multiple chat rooms for multiple clients. To ensure our application is secured, we will build our program using TCP protocol over UDP, as TCP is more favorable than UDP due to the reliable transfer rate that can be done. This application is developed using Python programming and its necessary libraries. The primary objective of this report is to present how this Multiple client-server chat program is developed and at the end demonstrate its result.

Introduction

Chatting is a method of using technology to bring people and ideas "together" despite geographical barriers. The technology has been available for years but the acceptance of it was quite recent. With the help of Socket programming, we are now able to achieve it. To define Socket programming, “it is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server”. It is presumed in socket programming that a client sends some information first and then the server or other clients respond to that information. This is not often the case in realistic contexts, it is not needed to deliver a message to someone to be able to receive one. Whenever it is sent, a recipient can readily receive a response, whereas transmitting and receiving must be introduced as independent rather than sequential processes.

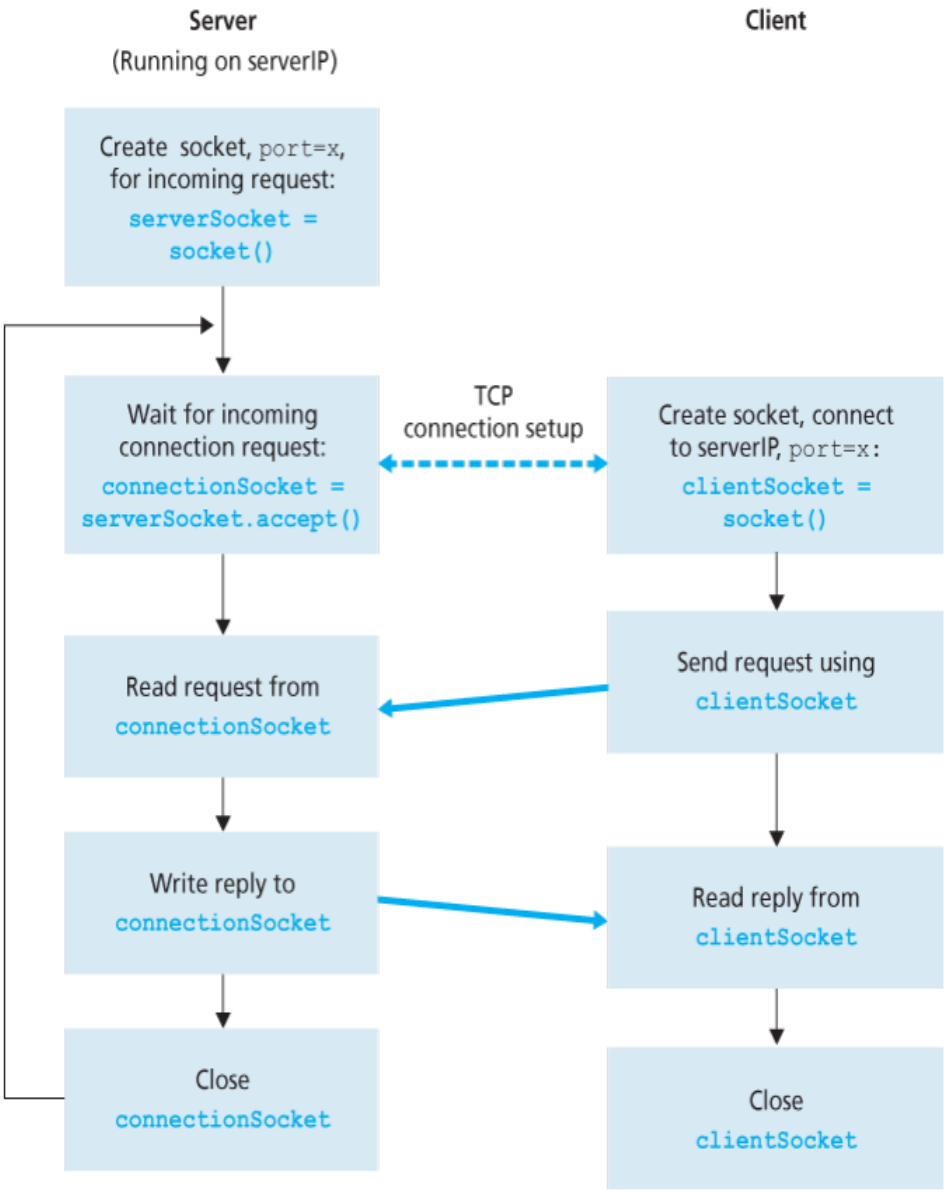
It is crucial to ensure the communication between clients to server and clients to client is secure. This leads us to use a protocol that would ensure security and make sure that the data is sent to the right destination, which in our case is TCP protocol. “A TCP (Transmission Control Protocol) works with the Internet Protocol (IP), which defines how computers send packets of data to each other. Together, TCP and IP are the basic rules defining the Internet. It is a connection-oriented protocol, and IP is the basic rule defining the Internet. It means that a connection is established and maintained until the application on programs at each end has finished exchanging messages”.

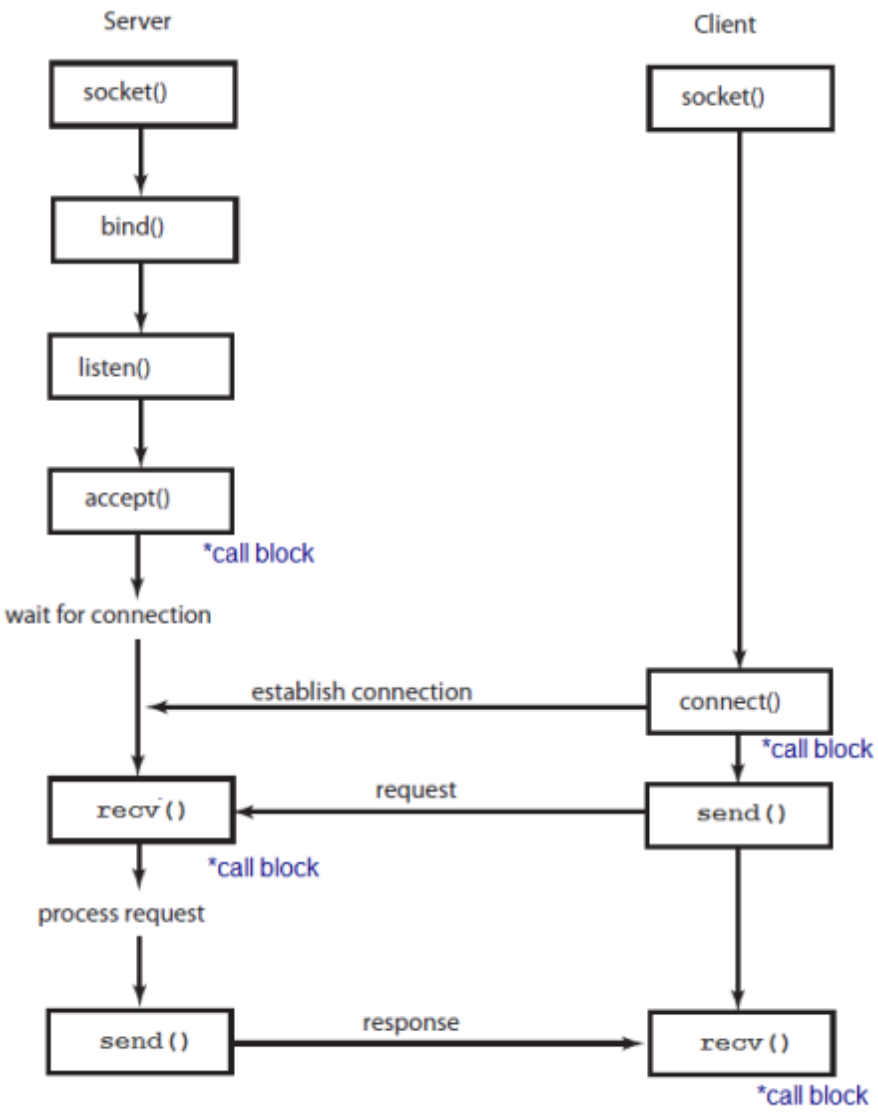
Our project is an example of a multiple-client chat server. It is made up of 2 applications: 1) the client application, which runs on the user's PC and 2) Server application, which runs on any PC on the network. To start chatting client should get connected to the server first. Below is a simple Client-Server model representation.



Design

Below is the representation of our client-server architecture.





Algorithm

The server-side process can be broken down into the following steps.

- Create a server socket using the given address family, socket type and protocol number.
- Bind the socket to address.
- Wait and Listen for the client to approach the server to make a connection.

- Once the TCP connection is established between server and client, receive the request (data) sent by client.
- Modify data and send response to the client.
- After sending response, close the server socket.

The Client-side process can be broken down into the following steps.

- Create a client socket
- Connect to server
- Then, send request to the server
- Next, receive reply (modified data) from the server.
- Finally, Close client socket.

Code \ Implementation

Server.py

```
# imports threading, socket, and sys libraries

import threading
import socket
import sys

# Creates the localhost IP and port, assigns to the server, and uses an empty
list for the clients and aliases variables
host = '127.0.0.1'
port = 59000
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen()
clients = []
aliases = []



# This function broadcasts messages to the client
```

```
def broadcast(message):
    for client in clients:
        client.send(message)

# This function handles clients' connections
def handle_client(client):
    while True:
        try:
            message = client.recv(1024)
            #exiting client
            print(f'Client: {client} and Client List: {clients}')
            if ".exit" in str(message):
                index = clients.index(client)
                clients.remove(client)
                client.close()
                alias = aliases[index]
                a_d = alias.decode('utf-8')
                msg_de = message.decode('utf-8')
                broadcast(f'{a_d} has left the chat room! with message
{msg_de}'.encode('utf-8'))
                aliases.remove(alias)
                break
            else:
                broadcast(message)
        except:

            print(f'Error Occurred! {sys.exc_info()[0]}')
            index = clients.index(client)
            clients.remove(client)
            client.close()
            alias = aliases[index]
            broadcast(f'{alias} has left the chat room!'.encode('utf-8'))
            aliases.remove(alias)
            break



# This function receives the clients connection
def receive():

    while True:

        print('Server is running and listening ...')
        client, address = server.accept()
        print(f'connection is established with {str(address)}')
        client.send('alias?'.encode('utf-8'))
        alias = client.recv(1024)
        aliases.append(alias)
        clients.append(client)
        a_d = alias.decode('utf-8')
        output = f'The alias of this client is {a_d}'.encode('utf-8')
        # This cleans up the unwanted b' in the output
        print(output.decode('utf-8'))
        broadcast(f'{a_d} has connected to the chat room'.encode('utf-8'))
        client.send('you are now connected!'.encode('utf-8'))
        thread = threading.Thread(target=handle_client, args=(client,))
        thread.start()
```

```
if __name__ == "__main__":

    receive()
```

Client.py

```
# imports threading, socket, and sys libraries

import threading
import socket


# This receives the user input
alias = input('Input username >>> ')

# here we created client socket
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(('127.0.0.1', 59000))



def client_receive():

    while True:

        try:
            message = client.recv(1024).decode('utf-8')

            if message == "alias?":

                client.send(alias.encode('utf-8'))

            else:
                print(message)

        except:

            a_d = str(alias)
            print(f'{a_d} has left the room!')
            client.close()
            break


# Here we check if a user exited the chat. If so, then the server removes
that specific user
def client_send():

    while True:

        message = f'{alias}: {input("")}'
```

```
        if ".exit" in str(message):

            client.send(message.encode('utf-8'))
            break

        else:

            client.send(message.encode('utf-8'))

    client.close()


# this section deals with threading
receive_thread = threading.Thread(target=client_receive)
receive_thread.start()

send_thread = threading.Thread(target=client_send)
send_thread.start()
```

Running the program

Note: To run the program, first, you will need to ensure that your computer has python installed.

If not you can do it using the link here https://www.python.org/downloads/

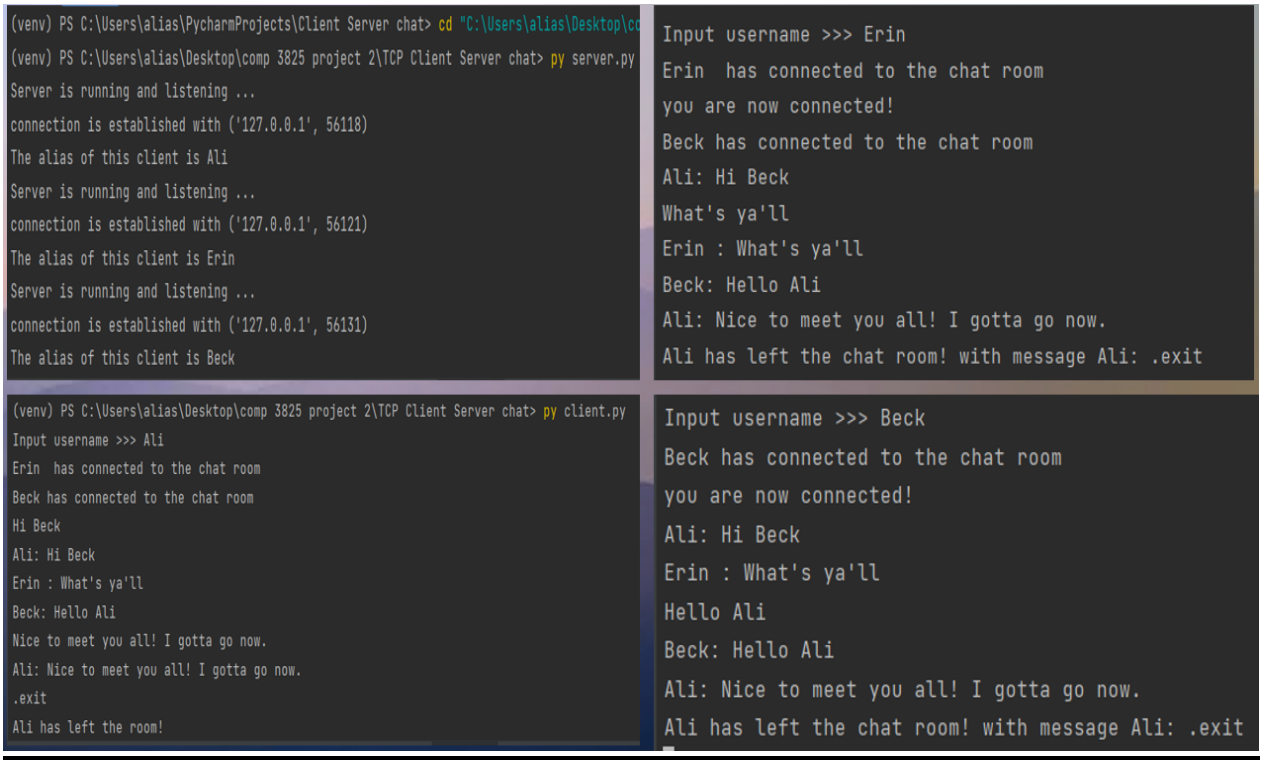
1. We will have to start and run our Server first. To do that open your PC's terminal and enter the following,

    python Server.py

2. Now, open another terminal window (the number of terminal windows should be open according to the number of clients you want to have chatting) to run the Client file, to start the conversation.

    python Client.py

Evaluation Result



As you can see in the screenshot above after running our program, our application is successfully running and showing the communication of multiple clients, which works like a group chat room.

Reference

GfG. “Socket Programming in C.” *GeeksforGeeks*, 9 Jan. 2024, www.geeksforgeeks.org/socket-programming cc/#:~:text=Socket%20programming%20is%20a%20way,reaches%20out%20to%20the%20server.

Hernández, María. “How to Simulate a TCP/UDP Client Using Netcat.” *Ubidots Blog*, 11 Mar. 2019, ubidots.com/blog/how-to-simulate-a-tcpudp-client-using-netcat.

“Multiple Client-server Communication(Chatroom) Using Socket Programming.” *Studocu*, www.studocu.com/in/document/dr-apj-abdul-kalam-technical-university/computer-networks-lab/exp8-multiple-client-server-communicationchatroom-using-socket-programming/29861392. Accessed 27 Nov. 2022.

Python Tutorial: Network Programming - Server and Client a : Basics - 2020. bogotobogo.com/python/python_network_programming_server_client.php.