Multiple Client - Server Communication Using Socket in Python

Dr. P. Punitha Ponmalar[1], G. Elakkiya[2]

[1]Associate Professor in Computer Science, Sri Meenakshi Govt Arts College for Women, Madurai-2, India

[2]II M. Sc Computer Science, Sri Meenakshi Govt Arts College for Women, Madurai-2, India

Abstract: ***This study's detailed description of socket programming in python's use to construct multiple client/server connections. To send messages over a network, sockets and the socket API are utilized. They provide a technique for interacting among processes (IPC). The client-server paradigm should be expanded to one where the several clients communicate with a server. There should be only one central server. This means, as using a distributed "peer-to-peer" processor. Before reaching the multiple client - server, present a client-server system that has just one client. The alternation system is more broadly applied then explained. Finally, the generic multi-client server is presented.***

Keywords: Socket programming in Python; Client- server Communication; Communicate message; Multiple Client - A Server Connection

1. Introduction

Multiple client - server software these days is no longer an uncommon notion in distributed computing and in computer networks [3]. The processes in this application must communicate with one another in order to share data among applications. The same machine or other machines connected by a network may house these processes. The multi clients and server apps are generally the two processes involved in this interprocess communication. Every client application makes a data request, to which the server application reacts by granting the request. The Multiple client - server architecture allows two remote entities to communicate. Internet applications frequently use HTTP for this reason [4].

One or more of the many alternatives, including file, signal, socket, shared memory, pipe, and message, can be used to facilitate interprocess communication. The main communication protocol used in the network infrastructure is IP/TCP (Internet Protocol/Transmission Control Protocol).

To exhibit the fundamental concepts of websockets, the researcher developed an application that would send the required contents of a shared message, share the server's time, and share the client's access to a file in the database [1].

2. Literature Review

Concepts and guiding ideas employed in socket programming in Python are used to assist this research investigation.

a) Multi-clients Server Communication

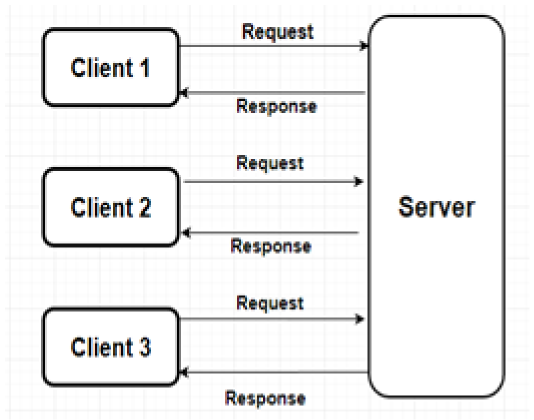


Figure 1: Multiple Client -Server Communication

According to its request-reply protocol, the multiple client/server paradigm in Fig. 1 above is used. Data would be sent primarily by the process of ensuring one or many data servers and one connection server, which further starts and ends connections. A peer-to-peer resource protocol is being used that also increases traffic on the network and has an effect on all network users. Wherever on the network, the data may be sent to the numerous customers, who can subsequently reassemble it or process it in another way [2].

b) Socket Programming

In a distributed computing environment, socket programming can be used to create client-server applications (composed of client and server programmes). A socket is a conduit for two different types of connectivity between running client and server applications. It features a strong two-computer communication system [5] [6].

Pair: Server and Socket Sockets are used to establish communication between two programmes and to carry out client and server functions [9].

c) Communication Protocol

For quick data transfer, a number of methods are available, including client-server protocol, single source, multiple mirror sources, and peer-to-peer file sharing. These methods call for extensive circumstances, demand and control mechanisms between the communication entities and are quite complicated. Datagram and stream communication channels are the two which are most frequently used during socket programming[3][7]. Just one server at any one moment in each of these methods initiates data transmission to the target client, who subsequently transfers the information to certain other peer clients. The client's TCP (Transfer Communication Protocol)advertised window size often determines the TCP transmission rate. In other methods, all customers are in communication with the main control unit [3].

3. Methodology

The TCP datagram-supported socket programming in Python PyCharm was used to create the application. The client-server programme is multifunctional. The client and server applications were simulated, displayed, and examined.

a) Proposed System

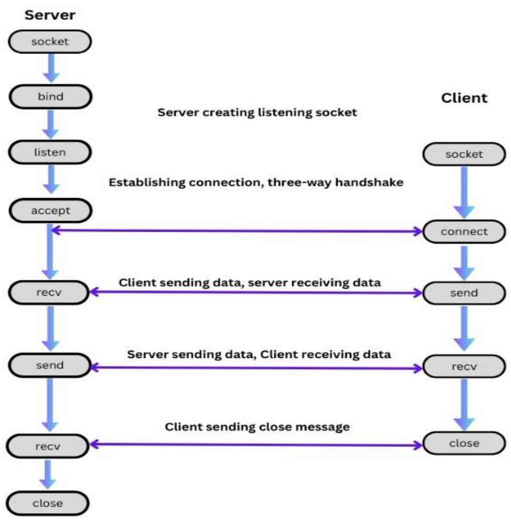

Server
socket
bind
listen
accept
recv
send
recv
close
Client
socket
connect
send
recv
close
Server creating listening socket
Establishing connection, three-way handshake
Client sending data, server receiving data
Server sending data, Client receiving data
Client sending close message

Figure 2: Socket Communication Server-Client

The server is represented by the lower left corner. The client is over there on the right. The server performs the following API calls to create a "listening" socket, starting in the top left-hand column: socket(), bind(), listen(), accept (). It keeps an ear out for client connections. The server calls accept() to accept or finish the relationship whenever a client joins. The customer calls. To begin the three-way communication and create a connection to the server, use connect(). The interaction phase is crucial because it guarantees that both ends of the connection can be reached through the network, or that the client can connect to the server and likewise. Just one host, client, or server may be able to connect to the other. A round part, which is in the center, is just where data is sent back and forth between the client and server via calls to send() and recv(). The server's and client's corresponding sockets are shut down at the bottom.

b) Application Concepts

In order to illustrate how socket programming should be used, this research research implemented a specific case when creating the multi-client server application. The research design and implement a server to communicate with each other client and respond to client requests. The research created and put into practise a server that interacted with clients and provided responses to their requests [5]. Also, there are numerous clients and one server involved in the programme, which has the following specifications:

- The client can ask the server, but the server has to be listening for that request.
- Both the client and the server must create a socket object. The server uses the method ***bind*** to bind a local address and an open port number, and the method***listen***to wait for a single incoming connection.
- Following that, it uses the ***accept()*** to accept the pending connection and replies to that connection with an encoded version of the string " ".
- The client, on the other hand, employs the ***connect*** method to establish a connection to the target host and port. As we are executing both processes in this instance on the same computer, the destination host will be the same as the source host.
- Data is obtained using the ***recv()***from the linked socket.
- The socket is shut down using the***close***technique by the client and the server.

c) Interface Design

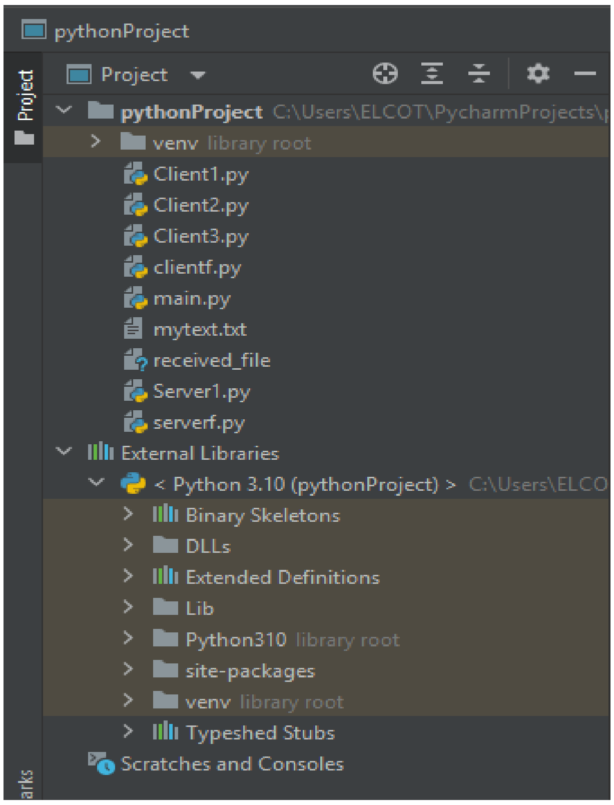
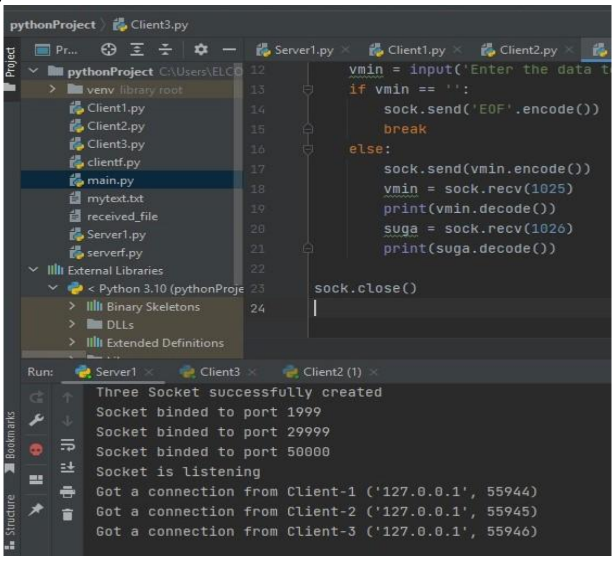

pythonProject
Project
pythonProject C:\Users\ELCOT\PycharmProjects\
venv library root
Client1.py
Client2.py
Client3.py
clientf.py
main.py
mytext.txt
received_file
Server1.py
serverf.py
External Libraries
< Python 3.10 (pythonProject) > C:\Users\ELCO
Binary Skeletons
DLLs
Extended Definitions
Lib
Python310 library root
site-packages
venv library root
Typeshed Stubs
Scratches and Consoles

Figure 3: Multiple Client Server Application creates Multiple Client Server Communication Project with Python Packages

Fig. 3 illustrates the programming approach used to create multi-client and server systems. The researchers employed a single project with the following Python classes: Client1, Client2, Client3, Clientf, mytext file, and Server1. In this instance, client and server software were coupled with one another to create a usable application that satisfied particular research objectives.

d) Main User Interface

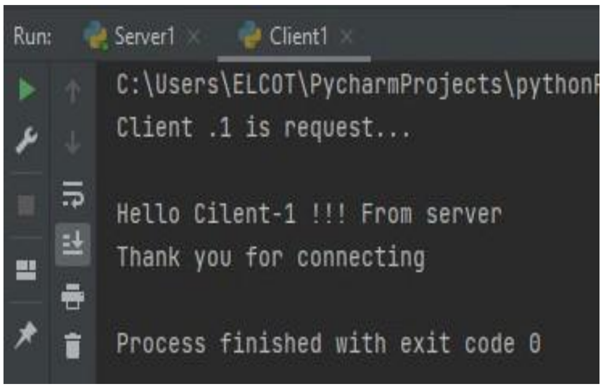


```
pythonProject > Client3.py

    vmin = input('Enter the data t
    if vmin == '':
        sock.send('EOF'.encode())
        break
    else:
        sock.send(vmin.encode())
        vmin = sock.recv(1025)
        print(vmin.decode())
        suga = sock.recv(1026)
        print(suga.decode())

sock.close()
```

```
Run: Server1  Client3  Client2 (1)
Three Socket successfully created
Socket binded to port 1999
Socket binded to port 29999
Socket binded to port 50000
Socket is listening
Got a connection from Client-1 ('127.0.0.1', 55944)
Got a connection from Client-2 ('127.0.0.1', 55945)
Got a connection from Client-3 ('127.0.0.1', 55946)
```

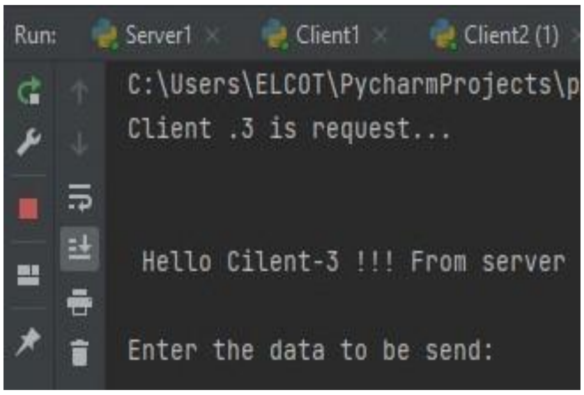
Figure 4: Server Output

Fig. 4 shows the application's process, in which all clients must connect to the server using their respective clients' individual port numbers and the same host name in order to get access.



```
Run: Server1  Client1
C:\Users\ELCOT\PycharmProjects\pythonP
Client .1 is request...

Hello Cilent-1 !!! From server
Thank you for connecting

Process finished with exit code 0
```

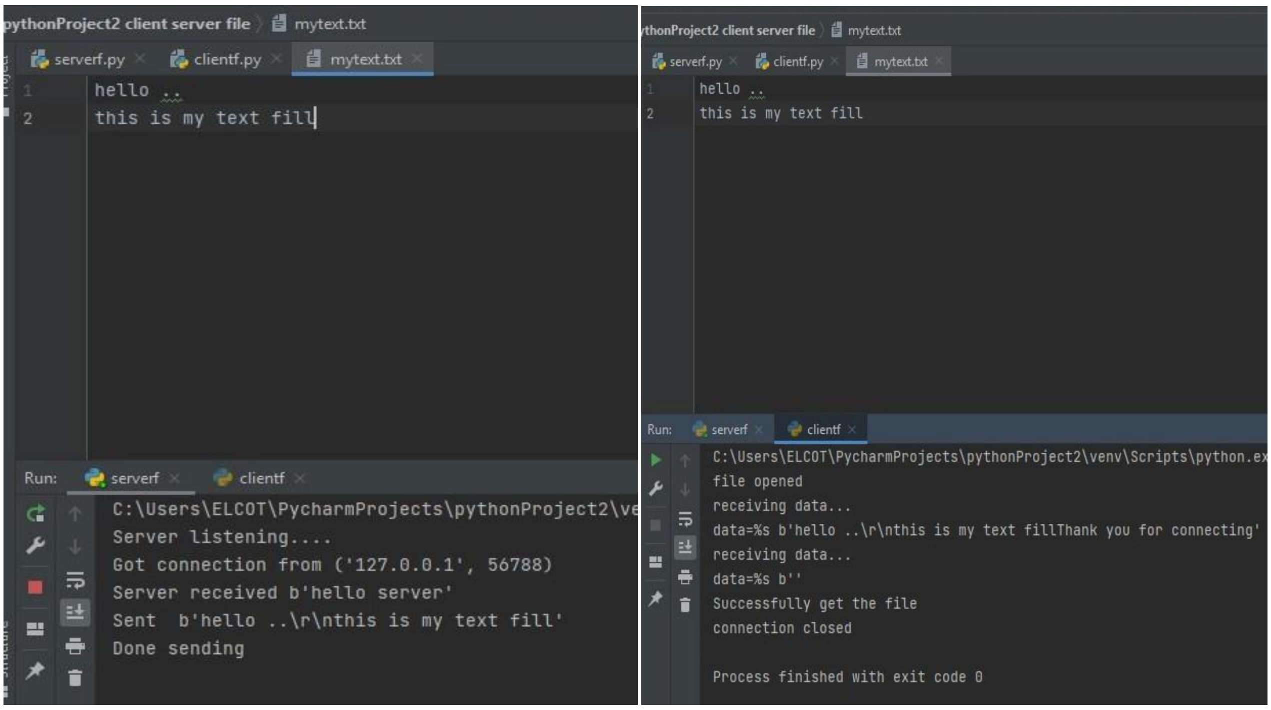
Figure 5: Client2 Connection



```
Run: Server1  Client1  Client2 (1)
C:\Users\ELCOT\PycharmProjects\p
Client .3 is request...

 Hello Cilent-3 !!! From server

Enter the data to be send:
```

Figure 6: Client1 Connection

Figures 5, and 6 depict the application's workflow. In order to get access, each client must establish a connection to the server using their unique port number and the same host name. The client sends the request, and the server responds with the client's request. Once the server has sent the clients any communicated messages or server time.



```
pythonProject2 client server file > mytext.txt
serverf.py  clientf.py  mytext.txt
hello ..
this is my text fill
```

```
Run: serverf  clientf
C:\Users\ELCOT\PycharmProjects\pythonProject2\ve
Server listening....
Got connection from ('127.0.0.1', 56788)
Server received b'hello server'
Sent  b'hello ..\r\nthis is my text fill'
Done sending
```

```
Run: serverf  clientf
C:\Users\ELCOT\PycharmProjects\pythonProject2\venv\Scripts\python.ex
file opened
receiving data...
data=%s b'hello ..\r\nthis is my text fillThank you for connecting'
receiving data...
data=%s b''
Successfully get the file
connection closed

Process finished with exit code 0
```

Figure 7: The client receives the file after the server sends the request file.

Fig. 7 depicts the application's workflow, in which clients must connect to the server using their unique client port numbers and the same host name in order to gain access. The client receives the file once the server sends the request file.

4. Conclusion and Future Work

In this research paper, the Multiple Client Server application using the socket programming approach is presented in depth as well as in an overview. Because Python (PyCharm) has a broad variety of classes and methods, we decided to use it to build this research. Also, researchers discovered that Python is an easy programming language to use in terms of programming style, principles, functions, datagrams, and socket programming. The created application may imitate a situation and show how socket programming is utilized and operates in a real-world setting. Thus, the researchers would want to suggest that the application be created in an object-oriented environment with an eye towards potential future expansion and development. In the interest of creating a future upgraded version of this programme, more capabilities may also be incorporated and implemented.

References

[1] Taneja, Sheetal, and Pratibha R. Gupta. "Python as a tool for web server application development."JIM 8I-International Journal of Information Communication and Computing Technology 2.1 (2014).

[2] Rawal, Bharat S., Lewis Berman, and Harold Ramcharan. "Multi-client/Multi-server split architecture."The International Conference on Information Networking 2013 (ICOIN). IEEE, 2013.

[3] IDG Communications (2017). Sockets Programming in Java World from IDG. [Online]. Availableat:https://www.javaworld.com/article/2077322/core-java/core-java-sockets-programming-in-java-atutorial.html. [Accessed 17-July-2017]

[4] Maata, Rolou Lyn & Cordova, Ronald &Sudramurthy, Balaji &Halibas, Alrence. Design andImplementation of Client-Server Based Application Using Socket Programming in a DistributedComputing Environment. 10.1109/ICCIC.2017.8524573. (2017).

[5] Jeff, C. “Sockets and Client-Server Communication”, Duke University, [Online] Available:http://db.cs.duke.edu/courses/spring06/cps196/slides/sockets.pdf [Accessed: 02-May-2017]

[6] Sinha, Alok. "Client-server computing." Communications of the ACM 35.7 (1992).

[7] Y. Li, L. Yang, and G. Yang, “Network-based coordinated motion control of large-scale transportation vehicles,” IEEE/ASME Trans. Mechatronics, Apr. 2007.

[8] M. Xue, C. Zhu, “The Socket Programming and Software Design for Communication Based onClient/Server”, IEEE Pacific-Asia Conference on Circuits, Communications and Systems, China, May 2009.

[9] Zelle, John M. "Python programming: an introduction to computer science." Pendiente de publicación. Borrador del libro disponible en http://mcsp. wartburg. edu/zelle/python (2002).