St. Francis Institute of Technology

Department of Computer Engineering

Academic Year: 2021-2022 **Semester: VIII**

Subject: Distributed Computing **Class/Branch/: BE/CMPNA**

Name :- Nithin Menezes **Roll Number: 56**

Experiment 1

Aim: Client Server Implementation Using Java socket programming.

Pre-requisites: Networking

Theory:

The endpoint in an interprocess communication is called a socket, or a network socket for disambiguation. Since most communication between computers is based on the Internet Protocol, an almost equivalent term is *Internet socket*. The data transmission between two sockets is organised by communications protocols, usually implemented in the operating system of the participating computers. Application programs write to and read from these sockets. Therefore, network programming is essentially socket programming.

Point-to-Point Communication

In a nutshell, a socket represents a single connection between exactly two pieces of software. More than two pieces of software can communicate in *client/server* or *distributed* systems (for example, many Web browsers can simultaneously communicate with a single Web server) but multiple sockets are required to do this. Socket-based software usually runs on two separate computers on the network, but sockets can also be used to communicate locally (*interprocess*) on a single computer.

Sockets are *bidirectional*, meaning that either side of the connection is capable of both sending and receiving data. Sometimes the one application that initiates communication is termed the *client* and the other application the *server*, but this terminology leads to confusion in nonclient/server systems and should generally be avoided.

Interface Types

Socket interfaces can be divided into three categories. Perhaps the most commonly-used type, the *stream* socket, implements "connection-oriented" semantics. Essentially, a "stream" requires that the two communicating parties first establish a socket connection, after which any data passed through that connection will be guaranteed to arrive in the same order in which it was sent.

Datagram sockets offer "connection-less" semantics. With datagrams, connections are implicit rather than explicit as with streams. Either party simply sends datagrams as needed and waits for the other to respond; messages can be lost in transmission or received out of order, but it is the application's responsibility and not the socket's to deal with these problems. Implementing datagram sockets can give some applications a performance boost and additional flexibility compared to using stream sockets, justifying their use in some situations.

The third type of socket -- the so-called *raw* socket -- bypasses the library's built-in support for standard protocols like TCP and UDP. Raw sockets are used for custom low-level protocol development.

Addresses and Ports

Today, sockets are typically used in conjunction with the Internet protocols -- Internet Protocol, Transmission Control Protocol, and User Datagram Protocol (UDP). Libraries implementing sockets for Internet Protocol use TCP for streams, UDP for datagrams, and IP itself for raw sockets.

To communicate over the Internet, IP socket libraries use the IP address to identify specific computers. Many parts of the Internet work with *naming services*, so that the users and socket programmers can work with computers by name(*e.g.*,"thiscomputer.compnetworking.about.com") instead of by address (*e.g.*, 208.185.127.40). Stream and datagram sockets also use IP port numbers to distinguish multiple applications from each other. For example, Web browsers on the Internet know to use port 80 as the default for socket communications with Web servers.

Program:

Server: # server.py

```
import time, socket,

sys
```

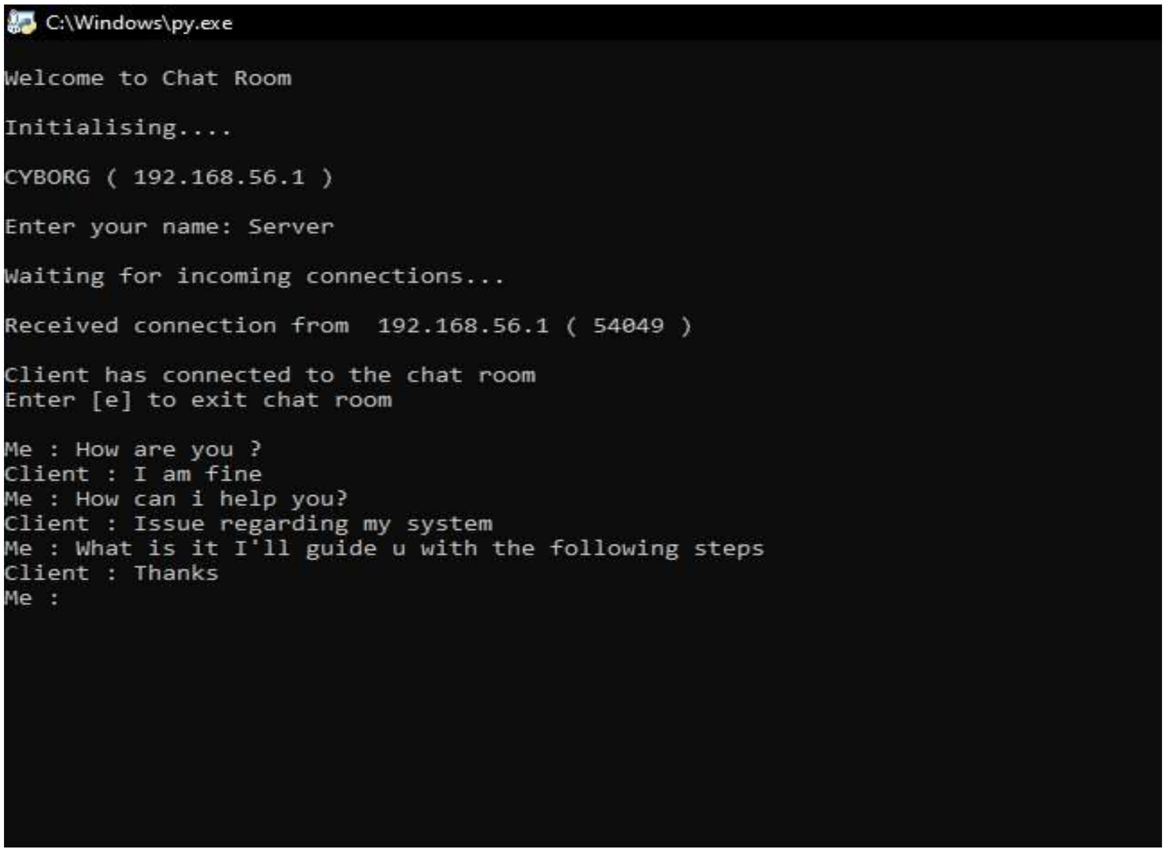
```
print("\nWelcome to Chat Room\n")

print("Initialising....\n")

time.sleep(1)



s = socket.socket()
host = socket.gethostname() ip

= socket.gethostbyname(host)

port = 1234

s.bind((host, port)) print(host, "(", ip,

")\n") name = input(str("Enter your

name: "))



s.listen(1) print("\nWaiting for incoming connections...\n")

conn, addr = s.accept() print("Received connection from ",

addr[0], "(", addr[1], ")\n")



s_name = conn.recv(1024) s_name = s_name.decode() print(s_name, "has

connected to the chat room\nEnter [e] to exit chat room\n")

conn.send(name.encode())



while True:

    message = input(str("Me : "))

if message == "[e]":
        message = "Left chat room!"

conn.send(message.encode())

        print("\n")
        break

    conn.send(message.encode())

message = conn.recv(1024)

message = message.decode()

print(s_name, ":", message)
```

<u>Client:</u> # client.py

```
import time, socket, sys



print("\nWelcome to Chat Room\n")

print("Initialising....\n") time.sleep(1)



s = socket.socket() shost =

socket.gethostname() ip =

socket.gethostbyname(shost)

print(shost, "(", ip, ")\n") host

= input(str("Enter server

address: ")) name =

input(str("\nEnter your name:

"))
```

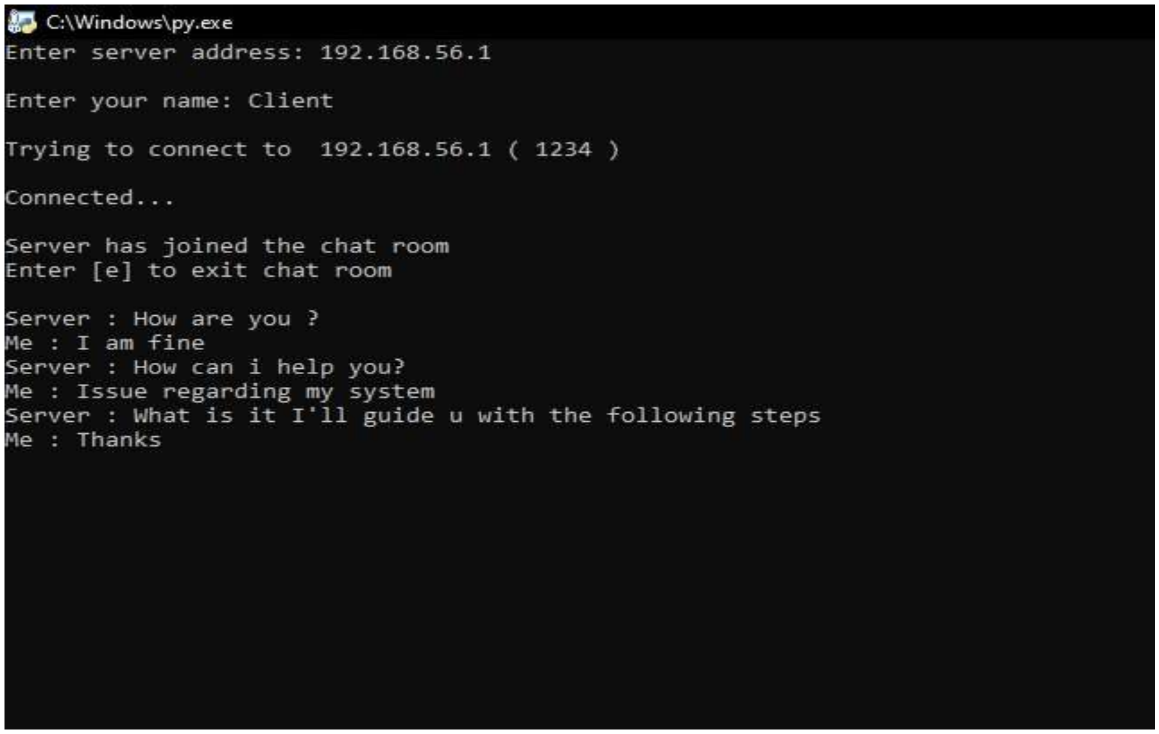
```
port = 1234 print("\nTrying to

connect to ", host, "(", port,

")\n") time.sleep(1)

s.connect((host, port)) print("Connected...\n")


s.send(name.encode()) s_name

= s.recv(1024) s_name =

s_name.decode() print(s_name,

"has joined the chat

room\nEnter [e] to exit chat

room\n")


while True:

    message = s.recv(1024)

message = message.decode()

print(s_name, ":", message)

message = input(str("Me : "))

if message == "[e]":

message = "Left chat room!"

        s.send(message.encode())         print("\n")         break

    s.send(message.encode())
```

Server Output:



```
C:\Windows\py.exe

Welcome to Chat Room

Initialising....

CYBORG ( 192.168.56.1 )

Enter your name: Server

Waiting for incoming connections...

Received connection from  192.168.56.1 ( 54049 )

Client has connected to the chat room
Enter [e] to exit chat room

Me : How are you ?
Client : I am fine
Me : How can i help you?
Client : Issue regarding my system
Me : What is it I'll guide u with the following steps
Client : Thanks
Me :
```

Client Output:



```
Enter server address: 192.168.56.1

Enter your name: Client

Trying to connect to  192.168.56.1 ( 1234 )

Connected...

Server has joined the chat room
Enter [e] to exit chat room

Server : How are you ?
Me : I am fine
Server : How can i help you?
Me : Issue regarding my system
Server : What is it I'll guide u with the following steps
Me : Thanks
```

Conclusion:

Thus socket programming which is used to communicate between computers is implemented successfully.