



**TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
THAPATHALI CAMPUS**

**A Lab Report
of
Distributed Systems
on
Simple Client Server Implementation using Java Socket Programming**

Submitted by:

Dipesh Bartaula

[THA076BCT015]

Submitted to:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

June 10, 2023

TITLE: SIMPLE CLIENT-SERVER IMPLEMENTATION USING JAVA SOCKET PROGRAMING

1. THEORY

A server is a computer or a program that provides services or resources to other computers or programs, known as clients. They are typically designed to be always available and reachable, running continuously to provide services. They can handle multiple client connections simultaneously or sequentially, depending on their design. Common types of servers include web servers, email servers, file servers, database servers, and game servers.

A client is a computer or a program that requests services or resources from a server. Clients send requests to the server, specifying the type of service they need or the data they want to access. Clients can be a part of various types of devices, such as personal computers, smartphones, tablets, or embedded systems. They interact with servers through various network protocols, such as HTTP for web browsing, SMTP for email, FTP for file transfer, etc.

The client-server model forms the backbone of networked applications and services, enabling distributed computing and resource sharing over a network. It facilitates the efficient utilization of resources by centralizing them on servers and providing access to clients whenever needed.

Java socket programming allows developers to create network applications that can communicate over TCP/IP networks using sockets. A socket is simply an endpoint for communications between the machines.

2. PROCEDURE

Java Socket programming is used for communication between the applications running on different JRE. Sockets provide a programming interface for network communication by encapsulating the underlying network protocols and allowing data to be sent and received between devices.

Brief procedure to the implementation of client-server:

a) Client side:

- The client code starts by creating a 'Socket' object and connecting it to the server's IP address and port number. We have use port number 9806 and host as localhost.
- 'BufferedReader' class is used to read string from user using 'readLine()' method.
- After reading string from user, 'PrintWriter' class is used to send message to the server using 'println()' method
- Then, response from server is received using 'BufferedReader' class and the response is printed to the console.

b) Server side:

- The server code starts by creating a 'ServerSocket' object, which listens for incoming connections on a specified port number same as that of client.

- The server enters a continuous loop to accept incoming client connections using the 'accept()' method of the 'ServerSocket' class. Once a client connects, it returns a 'Socket' object representing the client's socket.
- The server reads the client's request using the 'readLine()' method of the 'BufferedReader' class.
- After processing the request, the server sends a response to the client using the 'println()' method of the 'PrintWriter' class.

3. CODE

Client side:

```
import java.net.Socket;
import java.io.*;

public class client {

    public static void main(String[] args) {
        try
        {
            System.out.println("Client started");
            Socket soc = new Socket( host: "localhost", port: 9806);

            //Takes string as input from user via keyboard
            BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Enter a string:");
            String str = userInput.readLine();

            //Sending string to the server
            PrintWriter out = new PrintWriter(soc.getOutputStream(), autoFlush: true);
            out.println(str);

            //Reading and printing the data sent by the server
            BufferedReader in = new BufferedReader(new InputStreamReader(soc.getInputStream()));
            System.out.println(in.readLine());
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Server side:

```
import java.net.ServerSocket;
import java.net.Socket;
import java.io.*;

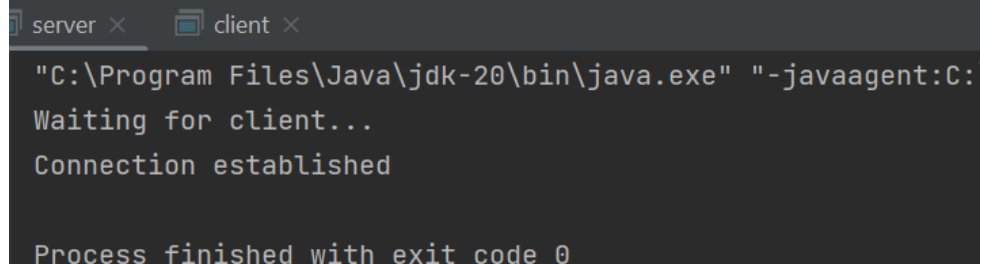
public class server {
    public static void main(String[] args) {
        try
        {
            System.out.println("Waiting for client...");
            ServerSocket ss = new ServerSocket(port: 9806);
            Socket soc = ss.accept();//establishes connection and waits for the client
            System.out.println("Connection established");

            //Read the string sent from client
            BufferedReader in= new BufferedReader(new InputStreamReader(soc.getInputStream()));
            String str = in.readLine();

            //Sending same string back to the client
            PrintWriter out = new PrintWriter(soc.getOutputStream(), autoFlush: true);
            out.println("Server responds: "+str);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

4. OUTPUT

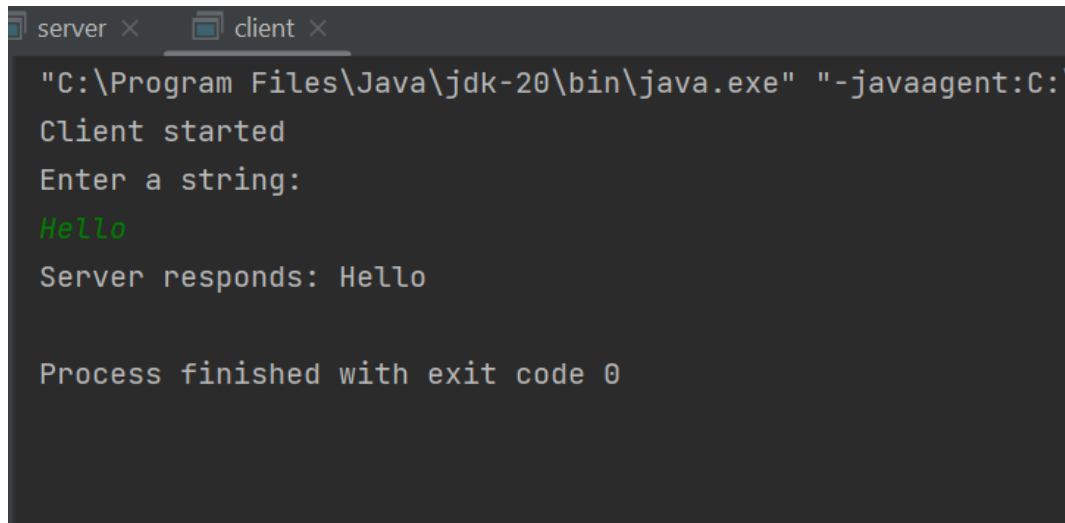
Server side:



```
server x client x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:
Waiting for client...
Connection established

Process finished with exit code 0
```

Client side:

A screenshot of a Java client application window titled 'client'. The window has a dark background and shows the following text: the command to run the client, 'Client started', a prompt 'Enter a string:', the user input 'Hello' in green, the server response 'Server responds: Hello', and the final status 'Process finished with exit code 0'.

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\
Client started
Enter a string:
Hello
Server responds: Hello

Process finished with exit code 0
```

5. CONCLUSION AND DISCUSSION

Above example demonstrates a simple client-server interaction using Java sockets. Here, both the server and client communicate using plain text messages. The server waits for a client to connect, reads the client's request, processes it (in this case, simply sending a predefined response). The client connects to the server, sends a request, waits for the server's response, and print the response.

This lab helps to gain knowledge on socket programming and mechanisms on how the client and server communicates.