# Experiment No - 4

**Aim:** To study client server communication.

***Theory:***

The **client–server model** is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web.

**CLIENT AND SERVER COMMUNICATION**

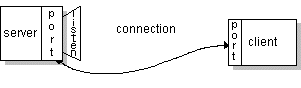
In general, a service is an abstraction of computer resources and a client does not have to be concerned with how the server performs while fulfilling the request and delivering the response. The client only has to understand the response based on the well-known application protocol, i.e. the content and the formatting of the data for the requested service. Clients and servers exchange messages in a request–response messaging pattern. The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication. To communicate, the computers must have a common language, and they must follow rules so that both the client and the server know what to expect. The language and rules of communication are defined in a communications protocol. All client-server protocols operate in the application layer. The application layer protocol defines the basic patterns of the dialogue. To formalize the data exchange even further, the server may implement an application programming interface (API). The API is an abstraction layer for accessing a service. By restricting communication to a specific content format, it facilitates parsing. By abstracting access, it facilitates cross-platform data exchange. A server may receive requests from many distinct clients in a short period of time. A computer can only perform a limited number of tasks at any moment, and relies on a scheduling system to prioritize incoming requests from clients to accommodate them. To prevent abuse and maximize availability, server software may limit the availability to clients. Denial of service attacks are designed to exploit a server's obligation to process requests by overloading it with excessive request rates.

**CLIENT AND SERVER SOCKET PROGRAMMING**

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request. On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

**Java Socket Programming**

Java Socket programming is used for communication between the applications running on different JRE. Java Socket programming can be connection-oriented or connectionless. Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connectionless socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

To make a process a TCP server, you need to follow the steps given below:

* Create a socket with the socket() system call.
* Bind the socket to an address using the bind() system call. For a server socket on the internet, an address consists of a port number on the host machine.
* Listen for connection with the listen() system call
* Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.
* Send and receive data using the read() and write() system calls.

**SERVER PROGRAM**

import java.io.\*;

import java.net.\*;

public class MyServer {

public static void main(String[] args) {

try{

ServerSocket ss = new ServerSocket(6666);

Socket s = ss.accept(); // establishes connection

DataInputStream dis = new DataInputStream(s.getInputStream());

String  str = (String)dis.readUTF();

System.out.println("message= "+str);

ss.close();

} catch(Exception e){System.out.println(e);}

}

}

**CLIENT PROGRAM**

import java.io.\*;

import java.net.\*;

public class MyClient {

public static void main(String[] args) {

try{

Socket s=new Socket("localhost",6666);

DataOutputStream dout = new DataOutputStream(s.getOutputStream());

dout.writeUTF("Hello Server");

dout.flush();

dout.close();

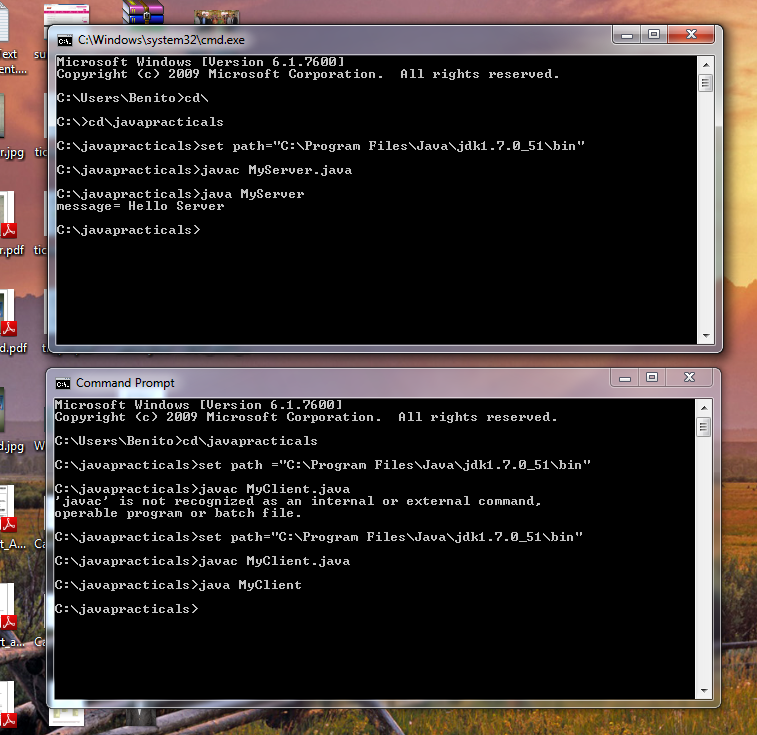
s.close();

} catch (Exception e) {System.out.println(e);}

}

}

***OUTPUT:***



***CONCLUSION:***

**Client-server** is a [software architecture](https://simple.wikipedia.org/wiki/Computer_architecture) model consisting of two parts, [client](https://simple.wikipedia.org/wiki/Client) systems and [server](https://simple.wikipedia.org/wiki/Server_(computing)) systems, both communicating over a [computer network](https://simple.wikipedia.org/wiki/Computer_network) or on the same [computer](https://simple.wikipedia.org/wiki/Computer). A client-server application is a [distributed system](https://simple.wikipedia.org/wiki/Distributed_computing) made up of both client and server software. Client server application provides a better way to share the workload. The client [process](https://simple.wikipedia.org/wiki/Process_(computing)) always initiates a connection to the server, while the server [process](https://simple.wikipedia.org/wiki/Process_(computing)) always waits for requests from any client.