Experiment No.1
Implement inter-process communication
Date of Performance: 16/02/23
Date of Submission: 23/02/23

Aim: To implement inter-process communication

Objective: Develop a program to implement inter-process communication using socket programming

Theory:-

URLs and URLConnections provide a relatively high-level mechanism for accessing resources on the Internet. Sometimes your programs require lower-level network communication, for example, when you want to write a client-server application.

In client-server applications, the server provides some service, such as processing database queries or sending out current stock prices. The client uses the service provided by the server, either displaying database query results to the user or making stock purchase recommendations to an investor. The communication that occurs between the client and the server must be reliable. That is, no data can be dropped and it must arrive on the client side in the same order in which the server sent it.

TCP provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection.

What Is a Socket?

A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.

Socket class
A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.
Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Code:

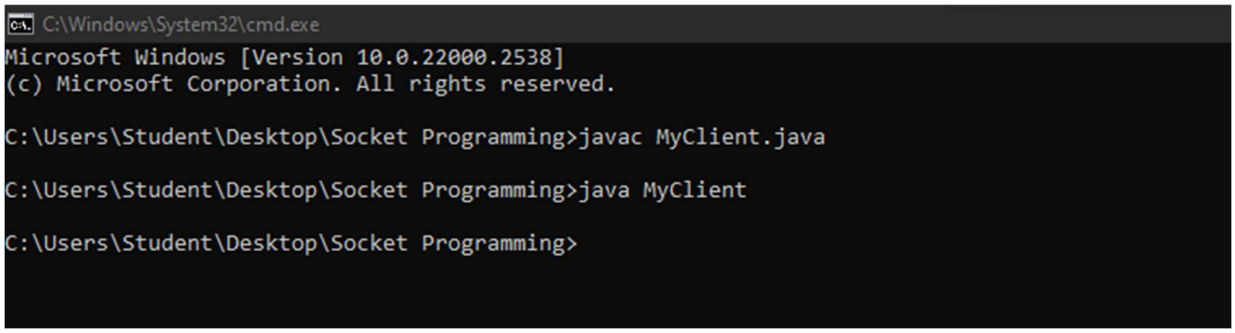
My Client.java

```
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("Welcome to DC Practicals");
dout.flush();

dout.close();
s.close();

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

MyServer.java

```
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);}
}
}
```

Output:



Conclusion: Implementing inter-process communication (IPC) using socket programming involves creating a server and client application where data can be exchanged between processes running on the same or different machines. Sockets provide a reliable and efficient means of communication, allowing processes to send and receive messages over a network. By establishing a connection between a server and client using sockets, data can be transmitted in both directions, facilitating effective communication between processes. Socket programming offers flexibility and scalability, enabling the development of distributed systems and networked applications. However, developers must handle error conditions and ensure proper synchronization to prevent issues such as data corruption or deadlock. Overall, socket programming is a powerful tool for enabling IPC and building robust distributed systems.