Experiment No.: 1

Aim: Write a program to perform addition of two numbers using Inter-Process Communication (IPC).

Theory:

In computer science, inter-process communication or inter process communication (IPC) refers specifically to the mechanisms an operating system provides to allow processes it manages to share data. Typically, applications can use IPC, categorized as clients and servers, where the client requests data and the server responds to client requests

- Mechanism for processes to communicate and to synchronize their actions.
- Message system – processes communicate with each other without resorting to shared variables.
- IPC facility provides two operations:
    - send(message) – message size fixed or variable
    - receive(message)
- If P and Q wish to communicate, they need to:
    - establish a communication link between them
    - exchange messages via send/receive
- Implementation of communication link
    - o physical (e.g., shared memory, hardware bus)
    - o logical (e.g., logical properties)

IPC Examples- TCP/IP sockets. Remote Procedure Calls (RPC), Remote Method Invocation (RMI), Message-passing Libraries for parallel computing, e.g. MPI, PVM ❍ Message-oriented Middleware.

Direct Communication: - Processes must name each other explicitly:

send (P, message) – send a message to process P

receive (Q, message) – receive a message from process Q

Only makes sense on a single computer unless distributed operating system that implements a

global process name space is being used. Properties of communication link: - Links are established automatically. A link is associated with exactly one pair of communicating processes. The link may be unidirectional, but is usually bidirectional.

Indirect Communication: - Messages are directed and received from mailboxes (also referred to as ports). Each mailbox has a unique id/address.

Primitives are defined as: send (A, message) – send a message to mailbox. A receive (A, message) – receive a message from mailbox A. The mailbox address A can be local or remote.

Operations: -create a new mailbox, send and receive messages through mailbox, destroy a mailbox

Issues in IPC: -

- Synchronous vs Asynchronous IPC
- Buffered vs unbuffered IPC
- Reliable vs unreliable (best effort)
- Ordered vs unordered
- Streams vs messages

Input:

1. Client Side:

```
import java.net.*;
import java.io.*;
import java.util.Scanner;


public class GreetingClient {
   public static void main(String[] args) {
      Scanner userInput = new Scanner(System.in);
      System.out.println("Enter Server Address: ");
      String serverName;
      serverName = userInput.next();
      System.out.println("Enter Port Number: ");
      String port;
      port = userInput.next();
      try {
         System.out.println("Connecting to " + serverName + " on port " + port);
         Socket client = new Socket(serverName, Integer.parseInt(port));
```

```
            System.out.println("Just connected to " + client.getRemoteSocketAddress());
            OutputStream outToServer = client.getOutputStream();
            DataOutputStream out = new DataOutputStream(outToServer);
            System.out.println("Enter a first number: ");
            //userInput.nextInt();
            Integer x = userInput.nextInt();
            System.out.println("Enter a second number: ");
            //  userInput.nextInt();
            Integer y = userInput.nextInt();
            //  System.out.println("hello");


            out.writeInt(x);
            out.writeInt(y);


            DataOutputStream os = new DataOutputStream(client.getOutputStream());
            BufferedReader is = new BufferedReader(new
InputStreamReader(client.getInputStream()));


            InputStream inFromServer = client.getInputStream();
            DataInputStream in = new DataInputStream(inFromServer);
            System.out.println("Server responds: " + in.read());
            client.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2. Server Side:

```
import java.net.*;
import java.io.*;
import java.util.Scanner;
```

```
public class GreetingServer extends Thread {
   private ServerSocket serverSocket;

   public GreetingServer(int port) throws IOException {
      serverSocket = new ServerSocket(port);
      serverSocket.setSoTimeout(100000);
   }
   public void run() {
      while (true) {
         try {
            System.out.println("Waiting for client on port " + serverSocket.getLocalPort() +
"...");
            Socket server = serverSocket.accept();
            System.out.println("Just connected to " + server.getRemoteSocketAddress());
            DataInputStream in = new DataInputStream(server.getInputStream());
            Integer x = in.readInt();
            System.out.println("Hellox");
            Integer y = in.readInt();
            System.out.println("Helloy");
            Integer sum = (x + y);
            DataOutputStream out = new DataOutputStream(server.getOutputStream());
            // System.out.println(sum);
            out.write(sum);
            server.close();
         } catch (SocketTimeoutException s) {
            System.out.println("Socket timed out!");
            break;
         } catch (IOException e) {
            e.printStackTrace();
            break;
         }
      }
```

```
    }

    public static void main(String[] args) {
        Scanner userInput = new Scanner(System.in);
        System.out.println("Please specify a port number (1~65535): ");
        String port;
        port = userInput.next();
        try {
            Thread t = new GreetingServer(Integer.parseInt(port));
            t.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

Output:

1. Client Side:

```
D:\Practical>java GreetingClient
Enter Server Address:
localhost
Enter Port Number:
5000
Connecting to localhost on port 5000
Just connected to localhost/127.0.0.1:5000
Enter a first number:
40
Enter a second number:
40
Server responds: 80

D:\Practical>
```

2. Server side:

```
D:\Practical>java GreetingServer
Please specify a port number (1~65535):
5000
Waiting for client on port 5000...
Just connected to /127.0.0.1:53753
Hellox
Helloy
Waiting for client on port 5000...
```

Conclusion:

We have seen the use of the primary IPC mechanisms with socket. Sockets are the most popular of all the IPC mechanisms for a simple reason: other mechanisms do not support communication between two machines. Sockets provide a full-duplex communication channel between processes that do not necessarily run on the same machine. The client process will send requests to the server, and the server process will send replies as an acknowledgment to the client requests.