LP V

Problem Statement 01 :

Implement multi threaded client/server Process communication using RMI

Create 4 Files

AddClient.java

AddServer.java

AddServerImpl.java

AddServerIntf.java

Imports for all

Import java.rmi.*

Import java.rmi.server.*

AddServerIntf :

It defines the interface and the function

Extends Remote

Public interface AddServerIntf extends Remote{

Public double add(double d1 , double d2) throws RemoteException ;

}

```
import java.rmi.*;

public interface AddServerIntf extends Remote{
    double add(double d1,double d2) throws RemoteException;
}
```

AddServerImpl.java

Extends UnicastRemoteObject

Implements AddServerIntf

```
import java.rmi.*;
import java.rmi.server.*;

public class AddServerImpl extends UnicastRemoteObject implements
AddServerIntf{
    public AddServerImpl() throws RemoteException{}

    public double add(double d1,double d2) throws RemoteException{
        return (d1+d2);
    }

}
```

AddServer.java

Import

Try

Create object of AddServerImpl

Catch

naming.bind(“AddServer”,object);

```
import java.rmi.*;

public abstract class AddServer {
    public static void main(String[] args){
        try{
            AddServerImpl addServerImpl = new
AddServerImpl();
            Naming.bind("AddServer",addServerImpl);
        }catch(Exception e){
            System.out.println("Exception "+e);
        }
    }
}
```

AddClient :

// main

String addServerURL = “rmi://”+args[o]+ “/AddServer”

AddServerIntf addServerIntf = (AddServerIntf) Naming.lookup(addServerURL)

```
import java.rmi.*;
import java.rmi.server.*;

public class AddClient{
    public static void main(String[] args) {
        try{
            String addServerURL = "rmi://"+args[0]+"/AddServer";
            AddServerIntf addServerIntf = (AddServerIntf)Naming.lookup(addServerURL);

            System.out.println("Enter the 1st Number");
            double d1 = Double.valueOf(args[1]).doubleValue();

            System.out.println("Enter the 1st Number");
            double d2 = Double.valueOf(args[1]).doubleValue();


            System.out.println("The sum is" + addServerIntf.add(d1, d2));

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

    }
}
```

How to Execute ?

Sudo apt-get install openjdk-8-jdk

Java –version
sudo update-java-alternatives --set /usr/lib/jvm/java-1.8.0
openjdk-amd64

javac *.java
rmic AddServerImpl
rmiregistry

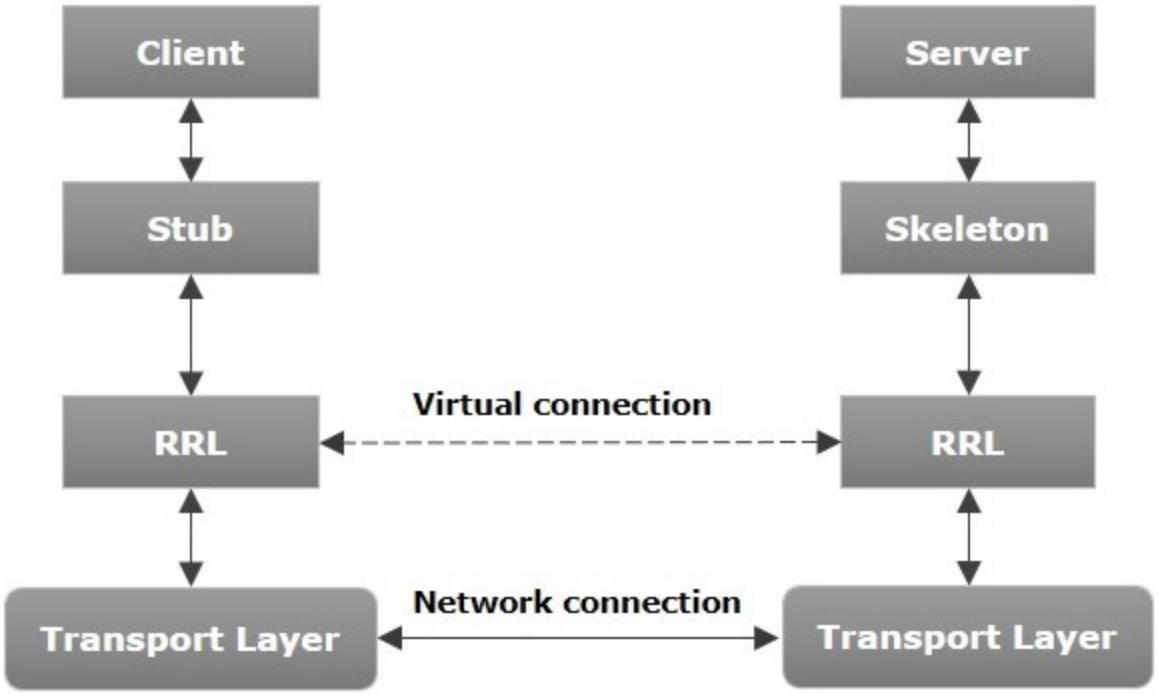
java AddServer

java AddClient "127.0.0.1" 5 10

Theory :

RMI (Remote Method Invocation) is a Java-based technology that enables communication and interaction between objects in a distributed system. It allows objects residing on different Java Virtual Machines (JVMs) to invoke methods on remote objects as if they were local objects, making it easier to build distributed applications in Java.

1. Remote Interface: RMI starts with the definition of a remote interface. The remote interface declares the methods that can be invoked remotely by client objects. Both the client and server need to have access to the interface's definition and implement it in their respective codebases.
2. Server Application: The server application provides the implementation of the remote interface. It creates an instance of the implementation class and registers it with the RMI runtime, making it available for remote invocations. The server typically runs on a separate JVM.
3. RMI Registry: The RMI registry acts as a central directory service for remote objects. It provides a way for clients to locate the remote objects they want to interact with. The server registers the remote object with the RMI registry, associating it with a unique name.
4. Client Application: The client application retrieves a reference to the remote object from the RMI registry using its unique name. This reference allows the client to invoke methods on the remote object as if it were a local object.
5. Method Invocation: When the client invokes a method on the remote object reference, RMI intercepts the call and performs the necessary network communication. It marshals the method parameters into a format that can be transmitted over the network and sends them to the server.
6. Remote Method Execution: The server receives the method invocation request, unmarshals the parameters, and executes the method on the actual object instance. If there is a return value, it is marshaled and sent back to the client through RMI.



Stub

A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.

Responsibilities :

1. Initiate remote calls
2. Marshall arguments to be sent
3. Inform the RRL to invoke the call
4. Unmarshall the return value
5. Inform the RRL the call is complete

Skeleton – This is the remote object which resides on the server side. stub communicates with this skeleton to pass requests to the remote object.

1. Unmarshall the incoming arguments from the client
2. Calling the actual remote object implementation
3. Marshaling the return value for the transport back to the client

RRL(Remote Reference Layer) –

It is the layer which manages the references made by the client to the remote object.

- When the client-side RRL receives the request, it invokes a method called invoke() of the object remoteRef. It passes the request to the RRL on the server side.

A remote object is an object whose method can be invoked from another JVM

Problem Statement 02 :

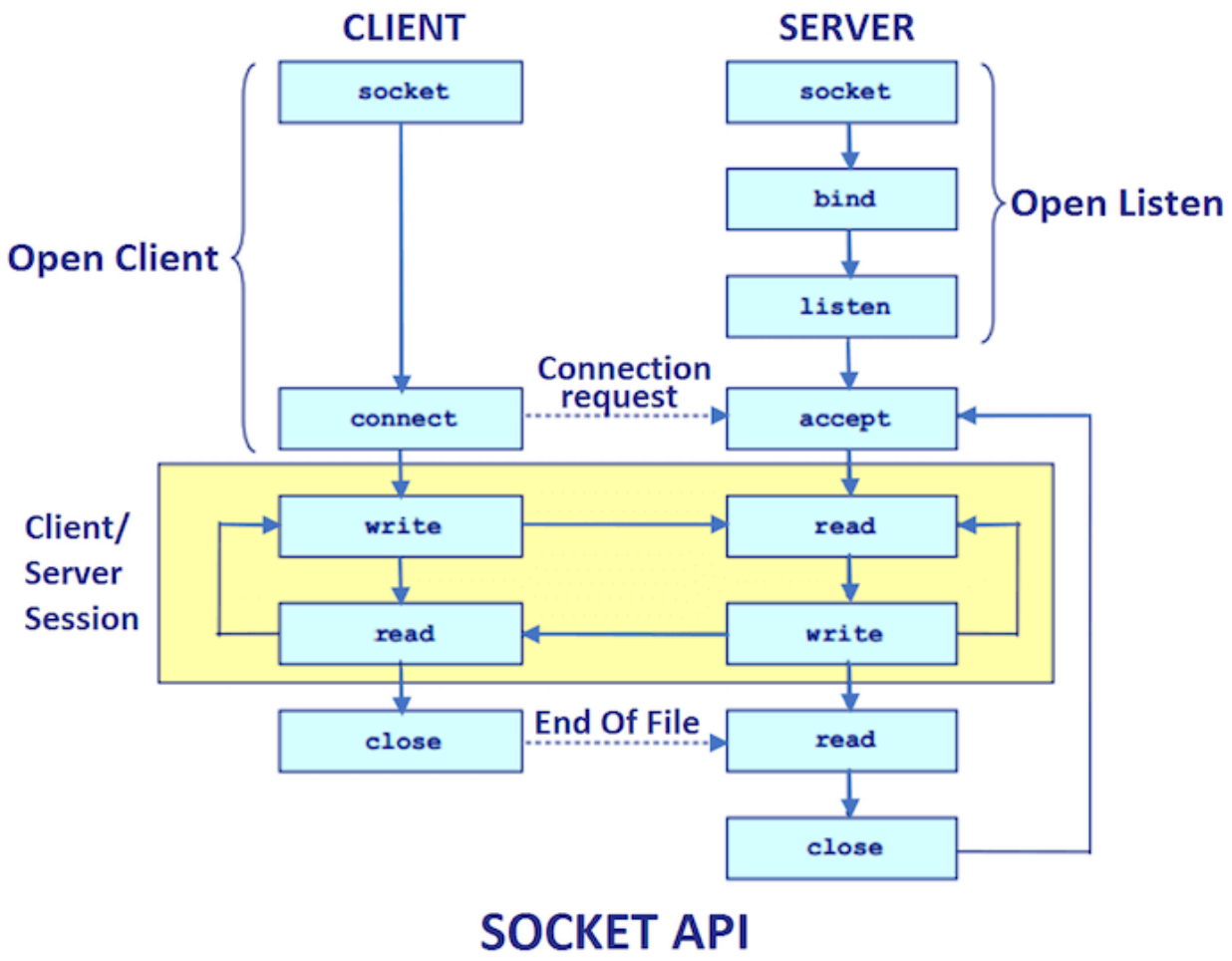
To develop any distributed application through implementing client communication programs based on Java Sockets

Theory :

session layer
Bidirectional

The client in socket programming must know two information:

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



Code :

Two Files

- Client

```
// client

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

public class client {
```

```
  public static void main(String[] args) throws Exception {
    try {
      Socket s = new Socket("127.0.0.1", 3333);
      DataInputStream din = new DataInputStream(s.getInputStream());
      DataOutputStream dout = new DataOutputStream(s.getOutputStream());
      Scanner sc = new Scanner(System.in);

      String str = "", str2 = "";
      while (!str.equals("stop")) {
        System.out.print("Client : ");
        str = sc.next();
        dout.writeUTF(str);
        str2 = din.readUTF();
        System.out.println("Server : " + str2);
      }
      dout.close();
      din.close();
      sc.close();
      s.close();
    } catch (Exception e) {
      System.out.println(e);
    }
  }
}
```

- Server

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

public class server {

  public static void main(String[] args) throws Exception {
    try {
      ServerSocket ss = new ServerSocket(3333, 0, null);
```

```
      Socket s = ss.accept();
      DataInputStream din = new DataInputStream(s.getInputStream());
      DataOutputStream dout = new DataOutputStream(s.getOutputStream());
      Scanner sc = new Scanner(System.in);

      String str = "", str2 = "";
      while (!str.equals("stop")) {
        str2 = din.readUTF();
        System.out.print("Client : " + str2);
        System.out.println("Server : ");
        str = sc.next();
        dout.writeUTF(str);
      }
      sc.close();
      din.close();
      dout.close();
      s.close();
      ss.close();
    } catch (Exception e) {
      System.out.println(e);
    }
  }
}
```

Run :

Javac *.java

Java server

Java client

Problem Statement 03 :

Develop any distributed application using CORBA to demonstrate object brokering. (Calculator operations).

CORBA (Common Object Request Broker Architecture) is a middleware technology that facilitates communication and interoperability between software components and systems. It is a standard defined by the Object Management Group (OMG) and provides a platform-independent mechanism for different software objects to communicate and interact with each other, regardless of their programming language or operating system.

1. Platform independent
2. Facilitates communication between devices

Common Object Request Broker Architecture

It is a collection of objects that isolates the requesters of a service from the providers by well defined encapsulating interface

CORBA differs from others as CORBA objects can

- Can run on any platform
- Can be located anywhere on the network
- Can be written in any language that has IDL mapping

CORBA is a client server system where client processes on the client machines can invoke operations on objects located on server machines

It was designed for heterogenous system

ORB

- Additional to RMI it has an extra layer of ORB
- The ORB manages the communication between the objects by invoking methods on remote objects and passing the parameters and return values between them.
- Each CORBA object is identified by an interface in a language called interface definition language
- ORB is a distributed service that implements the request of the remote object
- It locates the remote object on the network , communicates the request to the object waits for the results and communicates the result back to the client
- ORB implements location transparency
- In CORBA, an object request broker (ORB) acts as a middleware between the client and the server
- The ORB also provides services such as security, naming, and transaction management.

Interface and IDL :

This interface tells which methods the object exports and what parameters are required

CORBA provides a framework for objects in different languages and running on different operating systems to communicate with each other, regardless of where they are located on a network.

It uses an **interface definition language (IDL)** to describe the methods and attributes of objects, which are then used to generate code in different programming languages.. This allows developers to write distributed applications in different languages and running on different platforms, as long as they adhere to the CORBA standards.

IDL is used to define the interface .

Interface provides the methods that an object exports and specifies its parameters

One of the advantages of CORBA is its flexibility in terms of language and platform independence. It can be used to build distributed systems in different programming languages, such as C++, Java, Python, and others. Additionally, it can be used to integrate legacy systems and applications with modern ones, as long as they adhere to the CORBA standards.

Files :

1. ReverseModule.idl
2. ReverseImpl
3. ReverseClient
4. ReverseServer

problem3

ReverseModule.idl

```
module ReverseModule{
   interface Reverse{
      string     (in string str);
   };
};
```

//ReverseImpl

```
import ReverseModule.ReversePOA;

public class
```

```
extends ReversePOA {

  public String reverse_string(String name) {
    StringBuffer str = new StringBuffer(name);
    str.reverse();
    return ("Server Send " + str);
  }
}
-------------------------------------------------------------------------

//ReverseClient

import ReverseModule.*;
import java.util.Scanner;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.PortableServer.*;

class ReverseClient {

  public static void main(String args[]) {
    try {
      // initialize the ORB
      ORB orb = ORB.init(args, null);
      NamingContextExt ncRef = NamingContextExtHelper.narrow(
        orb.resolve_initial_references("NameService")
      );
      Reverse reverse = ReverseHelper.narrow(ncRef.resolve_str("Reverse"));

      System.out.println("Enter String : ");
      Scanner sc = new Scanner(System.in);
      String str = sc.next();
      String revStr = reverse.reverse_string(str);
      System.out.println(revStr);
    } catch (Exception e) {
      e.printStackTrace();
    }
```

```
 }
}
```

```
//ReverseServer

import ReverseModule.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.PortableServer.*;

public class ReverseServer {

  public static void main(String[] args) {
    try {
      ORB orb = ORB.init(args, null);

      POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootPOA.the_POAManager().activate();

      Reverse h_ref = ReverseHelper.narrow(
        rootPOA.servant_to_reference(new ReverseImpl())
      );

      NamingContextExt ncRef = NamingContextExtHelper.narrow(
        orb.resolve_initial_references("NameService")
      );
      NameComponent path[] = ncRef.to_name("Reverse");
      ncRef.rebind(path, h_ref);

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

Problem Statement 04 :

Problem Statement 05 :

Develop a distributed system, to find sum of N elements in an array by distributing N/n elements to n number of processors MPI or OpenMP. Demonstrate by displaying the intermediate sums calculated at different processors.

Theory :

MPI (Message Passing Interface) is a standardized communication protocol and library that allows parallel computing across multiple processors or nodes in a distributed computing environment. It enables communication and coordination among different processes running on separate nodes, typically used for high-performance computing (HPC) applications

Scatter and Gather are two fundamental collective communication operations in MPI. They are often used in parallel computations where data needs to be distributed across processes and then gathered back for further processing.

1. Scatter:
   The Scatter operation allows a root process to distribute a portion of data to all other processes in the communicator. It divides the data into equal-sized chunks and sends each chunk to a different process.

In Java, you can use the MPI Scatter operation by calling the `MPI.COMM_WORLD.scatter()` method. The method takes the following parameters:

- sendbuf: The data array to be sent from the root process. In the root process, this array contains the entire data, while in other processes, it can be an empty array.
- sendoffset: The offset in the send buffer from where the data starts.
- sendcount: The number of elements to be sent to each process.
- sendtype: The datatype of the elements in the send buffer.
- recvbuf: The receive buffer in each process. It will store the received portion of data.
- recvoffset: The offset in the receive buffer where the received data will be stored.
- recvcount: The number of elements to be received by each process.
- recvtype: The datatype of the elements in the receive buffer.
- root: The rank of the root process that distributes the data.

2. Gather:
   The Gather operation collects data from all processes in a communicator and gathers it into the root process. It is the reverse of Scatter.

In Java, you can use the MPI Gather operation by calling the **`MPI.COMM_WORLD.gather()`** method. The method takes similar parameters to Scatter, but the roles of sendbuf and recvbuf are reversed. The root process will receive data from all other processes and store it in the receive buffer.

After calling the Gather method, the root process will have the gathered data in its receive buffer, while other processes will have empty receive buffers.

Both Scatter and Gather operations are collective operations, meaning all processes in the communicator must participate in the operation. These operations provide an efficient way to distribute and gather data in parallel computations, allowing for better performance and scalability in distributed computing scenarios.

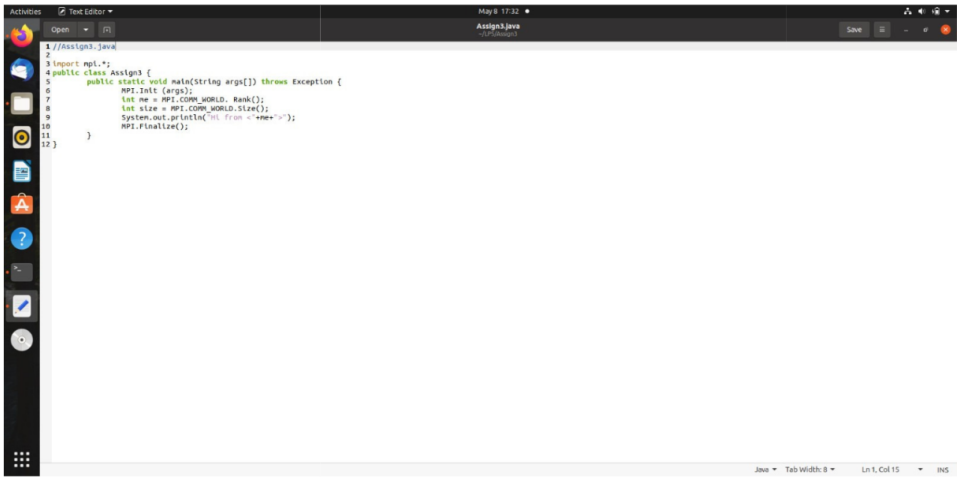
Code :

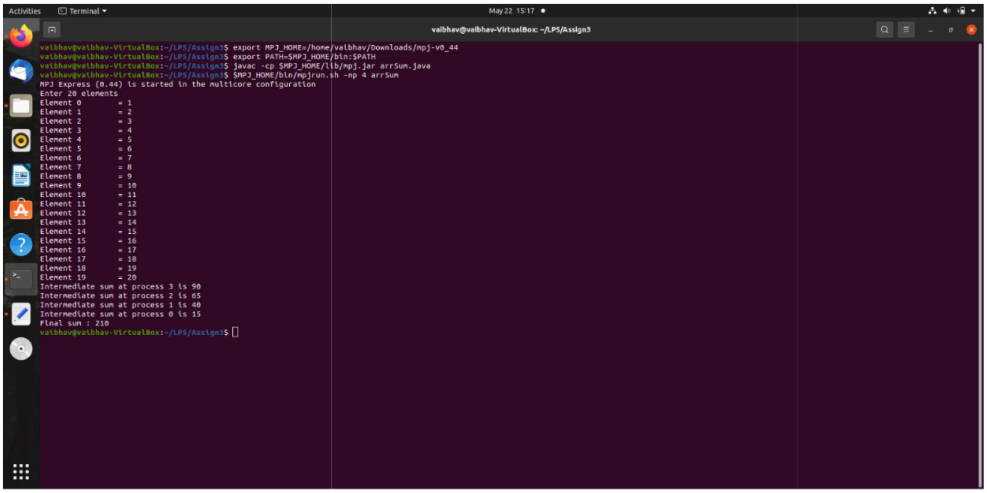
```
problem5
--------------------------------------------------------------------------------
import mpi.MPI;

public class arrSum {

  public static void main(String[] args) {
    MPI.Init(args);
    int rank = MPI.COMM_WORLD.Rank();
    int size = MPI.COMM_WORLD.Size();
    int unitsize = 5, root = 0;
    int send_buffer[] = new int[unitsize * size];
    int recieve_buffer[] = new int[unitsize];
    int new_receiver_buffer[] = new int[size];
    if (rank == root) {
      int total = unitsize * size;
```

```
      for (int i = 0; i < total; i++) send_buffer[i] = i + 1;
    }
    MPI.COMM_WORLD.Scatter(
      send_buffer,
      0,
      unitsize,
      MPI.INT,
      recieve_buffer,
      0,
      unitsize,
      MPI.INT,
      root
    );
    for (int i = 1; i < unitsize; i++) recieve_buffer[0] += recieve_buffer[i];
    System.out.println(
      "Intermediate sum at  " + rank + " is " + recieve_buffer[0]
    );
    MPI.COMM_WORLD.Gather(
      recieve_buffer,
      0,
      1,
      MPI.INT,
      new_receiver_buffer,
      0,
      1,
      MPI.INT,
      root
    );
    if (rank == root) {
      int total = 0;
      for (int i = 0; i < size; i++) total += new_receiver_buffer[i];
      System.out.println("Total sum is " + total);
    }
    MPI.Finalize();
  }
}
```

What are the things to remember in here

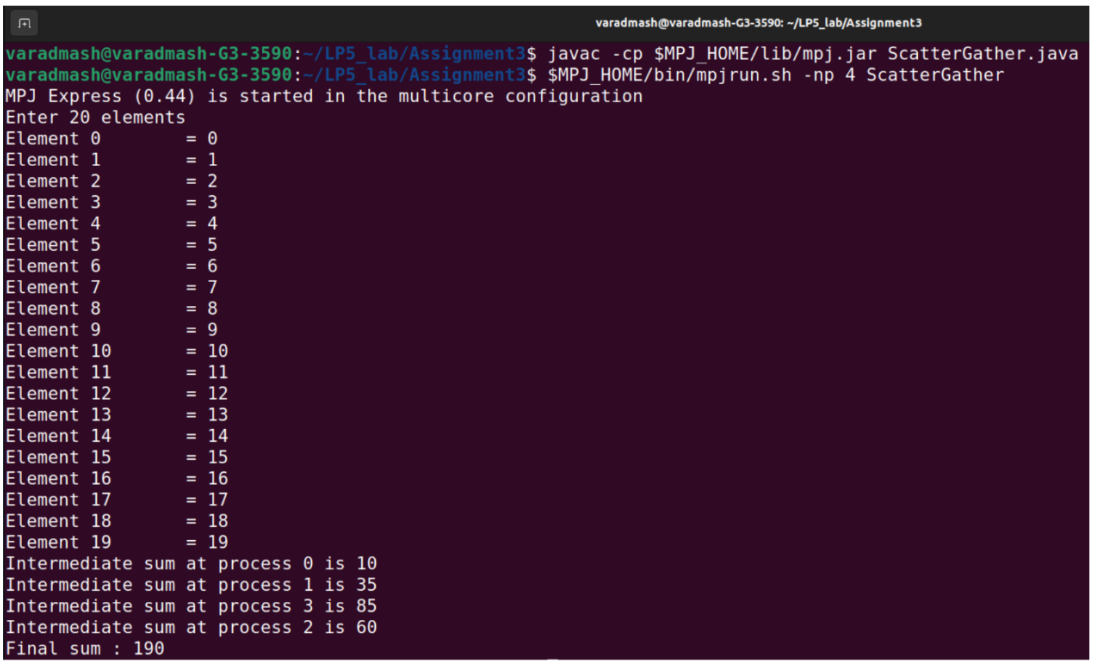
1. MPI.Init(args)
2. Int rank = MPI.COMM_WORLD.Rank();
3. Int size = MPI.COMM_WORLD.Size();
4. int uintSize = 5
5. Root = 0;
6. Send_buffer[uintSize*size)
7. recieve_buffer[uintSize]
8. New_recive_buffer[size]
9. if(root==rank)
10. Total_elements = uintSize*size
11. Take input of array in send_buffer[i] = i+1;
12. Scatter data to the processes
13. MPI.Scatter(send_buffer,0,uintSize,MPI.INT,recieve_buffer, 0,uintSize,MPI.INT,root)
14. Aggregate the output at the recieve_buffer[0];
15. Intermediate sum at the process
16. MPI.COMM_WORLD.Gather(recive_buffer,0,1,MPI.INT,recieve_buffer,0,1 ,MPI.INT,root)
17. Calculate total_sum for (i = 0 to size) total_sum+=new_recieve_buffer[i]
18. MPI.finalize()


Activities
Text Editor
May 8 17:32
Open
Assign3.java
~/LPS/Assign3
Save
1 //Assign3.java
2
3 import mpi.*;
4 public class Assign3 {
5 public static void main(String args[]) throws Exception {
6 MPI.Init (args);
7 int me = MPI.COMM_WORLD. Rank();
8 int size = MPI.COMM_WORLD.Size();
9 System.out.println("Hi from <" +me+">");
10 MPI.Finalize();
11 }
12 }
Java
Tab Width: 8
Ln 1, Col 15
INS


Activities
Terminal
May 22 15:17
valbhav@valbhav-VirtualBox: ~/LPS/Assign3
valbhav@valbhav-VirtualBox:~/LPS/Assign3$ export MPJ_HOME=/home/valbhav/Downloads/mpj-v0_44
valbhav@valbhav-VirtualBox:~/LPS/Assign3$ export PATH=$MPJ_HOME/bin:$PATH
valbhav@valbhav-VirtualBox:~/LPS/Assign3$ javac -cp $MPJ_HOME/lib/mpj.jar arrSum.java
valbhav@valbhav-VirtualBox:~/LPS/Assign3$ $MPJ_HOME/bin/mpjrun.sh -np 4 arrSum
MPJ Express (0.44) is started in the multicore configuration
Enter 20 elements
Element 0 = 1
Element 1 = 2
Element 2 = 3
Element 3 = 4
Element 4 = 5
Element 5 = 6
Element 6 = 7
Element 7 = 8
Element 8 = 9
Element 9 = 10
Element 10 = 11
Element 11 = 12
Element 12 = 13
Element 13 = 14
Element 14 = 15
Element 15 = 16
Element 16 = 17
Element 17 = 18
Element 18 = 19
Element 19 = 20
Intermediate sum at process 3 is 90
Intermediate sum at process 2 is 65
Intermediate sum at process 1 is 40
Intermediate sum at process 0 is 15
Final sun : 210
valbhav@valbhav-VirtualBox:~/LPS/Assign3$

Output:

1. Compilation and Execution



1. Compile with the mpj.jar file in the lib
2. Location home / lib / mpj.jar
3. javac -cp “location of mpj.jar” arrSum.java
4. home/bin/mpjrun.sh -np 4 arrSum

Problem Statement 06 :

Implement Berkeley algorithm for clock synchronization

Theory :

Berkeley Algorithm :

The goal of the algorithm is to synchronize the clocks of multiple computers in a network, even if there are significant variations in clock drift rates among the computers.

The basic idea of the Berkeley algorithm is as follows:

1. One computer is elected as the time server or coordinator. This can be done through a voting process or a predetermined selection.
2. The time server periodically polls the other computers in the network for their local clock values.
3. Upon receiving the responses from the other computers, the time server calculates the average clock value, excluding outliers that deviate significantly from the average.

4. The time server then sends the adjusted time value to each computer, which adjusts its local clock to match the new value.
5. The process is repeated periodically to continuously synchronize the clocks of all computers in the network.

By using this algorithm, the clocks of the computers in the network can be brought into reasonable agreement, reducing the effects of clock drift and maintaining a synchronized notion of time across the distributed system.

Code

This is an easy experiment :

Two files : client.java server.java

Client :

1. Extends thread class
2. while(true){
3. Create a socket sc = new Socket("localhost",port)
4. inputStream
5. OutputStream
6. Int clientTime = System.currentTimeMillis();
7. outPutStream.writeLong(clientTime)
8. Long ack = inputStream.readUTF()
9. print(ack)
10. Long avg = inputStream.readLong()
11. Print currentTIme
12. Print average time
13. Close all

14. Thread.sleep(1000);

```
//client.java
import java.io.*;
import java.net.*;
import java.util.*;

public class client extends Thread {

  public static void main(String[] args) {
    try {
      // Connect to the server
      while (true) {
        Socket socket = new Socket("localhost", 12345);
        DataInputStream inputStream = new DataInputStream(
          socket.getInputStream()
        );
        DataOutputStream outputStream = new DataOutputStream(
          socket.getOutputStream()
        );

        long clientTime = System.currentTimeMillis();
        outputStream.writeLong(clientTime);
        String acknowledgment = inputStream.readUTF();

        System.out.println(acknowledgment);

        long averageTime = inputStream.readLong();
        System.out.println("------------------------------------");
        System.out.println("Current time : " + new Date(clientTime));
        System.out.println("Adjusted time : " + new
Date(averageTime));
        System.out.println("------------------------------------");
```

```
        inputStream.close();
        outputStream.close();
        socket.close();

        Thread.sleep(10000);
      }
    } catch (Exception e) {
      System.out.println(e);
    }
  }
}
```

Server.java

1. Does not extend thread class
2. Create a serverSocket
3. Create an arrayList<Long> clientTimes
4. clientTimes.add(System.currentTimeMillis());
5. while(client.times.size()<1000){
    a. Socket sc = serverSocket.accept()
    b. inputStream
    c. OutputStream
    d. clientTime = inputStream.readLong();
    e. Add it to the ArrayList
    f. outstream.writeUTF("This is the time received by the server" + clientTime)
    g. Calculate the sum
    h. Calculate avg

i. outputStream.writeLong(avg);

j. Close all streams

```
//server.java
import java.io.*;
import java.net.*;
import java.util.*;

public class server extends Thread {

  public static void main(String[] args) {
    try {
      ServerSocket serverSocket = new ServerSocket(12345);
      List<Long> clientTimes = new ArrayList<>();
      clientTimes.add(System.currentTimeMillis());
      System.out.println("Server is running. Waiting for clients...");
      while (clientTimes.size() < 1000) {
        System.out.println("------------------------------------");
        Socket clientSocket = serverSocket.accept();
        DataInputStream inputStream = new DataInputStream(
          clientSocket.getInputStream()
        );
        DataOutputStream outputStream = new DataOutputStream(
          clientSocket.getOutputStream()
        );
        System.out.println("listening to client timing ");

        long clientTime = inputStream.readLong();
        clientTimes.add(clientTime);

        outputStream.writeUTF(
          "Time received by the server." + new Date(clientTime)
        );
        long sum = 0, n = clientTimes.size();
        for (int i = 0; i < n; i++) {
          sum += clientTimes.get(i);
        }

        System.out.println("Number of client to be synchronized " + n);
```

```
        System.out.println("-----------------------------------");

        long averageTime = sum / clientTimes.size();
        outputStream.writeLong(averageTime);
        inputStream.close();
        outputStream.close();
        clientSocket.close();
      }
      serverSocket.close();
    } catch (Exception e) {
      System.out.println(e);
    }
  }
}
----------------------------------------------------------------------
```

Problem Statement 07 :

Implement token ring based mutual exclusion algorithm

Token ring based mutual exclusion algorithm is a distributed algorithm used to coordinate access to shared resources in a network of interconnected nodes organized in a ring topology. The algorithm ensures that only one node at a time has exclusive access to the shared resource, preventing conflicts and maintaining consistency.

The algorithm operates by passing a token, which is a special message or control packet, around the ring. The token circulates through the nodes in a predetermined order, and a node can access the shared resource only when it possesses the token. When a node needs to access the resource, it waits until it receives the token, indicating

that it is its turn to use the resource. After using the resource, the node releases the token and passes it to the next node in the ring.

The token ring based mutual exclusion algorithm can be implemented using various approaches, but generally, it follows these steps:

1. Initialization: Each node in the ring is assigned a unique identifier, and the initial token is assigned to a particular node.
2. Requesting access: When a node wants to access the shared resource, it waits until it receives the token from the preceding node.
3. Critical section execution: Once a node receives the token, it enters the critical section and performs the desired operations on the shared resource.
4. Releasing access: After completing its critical section, the node releases the token and passes it to the next node in the ring, allowing another node to access the resource.

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


public class ring{
    public static void main(String[] args) {
        int sender = 0;
        int receiver = 0;
        int token = 0;

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter number of process : ");
        int n = sc.nextInt();
        for(int i = 0;i<n;i++){
            System.out.print(" p"+i);
        }
        System.out.println(" p"+0);

        while(true){
```

```
            System.out.println("-----------------");
            System.out.println("Enter Sender : ");
            sender = sc.nextInt();
            System.out.println("Enter Receiver  :");
            receiver = sc.nextInt();
            System.out.println("Token passes : ");
            for(int i = token;i!=sender;i = (i+1)%n){
                System.out.print(i+" -->");
            }
            System.out.println();
            System.out.println(sender);
            System.out.println("Enter data to be sent : ");
            String data = sc.next();


            for(int i = sender;i!=receiver;i = (i+1)%n){
                System.out.println(data + " forwarded by "+i);
            }

            System.out.println(data + " received at " + receiver);
            token = sender;
            System.out.println("--------------------------");
        }
    }
}
```

Problem Statement 08 :

Bully Algorithm:

The Bully Algorithm is a leader election algorithm where nodes with higher IDs have more priority in becoming the leader.

Ring Algorithm:

The Ring Algorithm is a leader election algorithm where nodes form a logical ring and pass a token to elect a leader.

The **bully** algorithm is a type of **Election algorithm** which is mainly used for choosing a coordinate. In a distributed system, we need some election algorithms such as **bully** and **ring** to get a coordinator that performs functions needed by other processes.

Election algorithms select a single process from the processes that act as coordinator. A new process is selected when the selected coordinator process crashes due to some reasons. In order to determine the position where the new copy of coordinator should be restarted, the election algorithms are used.

The **Bully** election algorithm is as follows:

Let's assume that **P** is a process that sends a message to the coordinator.

1. It will assume that the coordinator process is failed when it doesn't receive any response from the coordinator within the time interval **T**.
2. An election message will be sent to all the active processes by process **P** along with the highest priority number.
3. If it will not receive any response within the time interval T, the current process P elects itself as a coordinator.
4. After selecting itself as a coordinator, it again sends a message that process **P** is elected as their new coordinator to all the processes having lower priority.
5. If process **P** will receive any response from another process **Q** within time **T:**
    a. It again waits for time T to receive another response, i.e., it has been elected as coordinator from process **Q**.

b. If it doesn't receive any response within time **T**, it is assumed to have failed, and the algorithm is restarted.

The Bully Algorithm is a leader election algorithm used in distributed systems, where a group of interconnected nodes or processes need to elect a leader to perform certain tasks or make decisions on behalf of the group. The algorithm ensures that the most capable node, typically the one with the highest ID, becomes the leader.

Here's a step-by-step explanation of the Bully Algorithm:

1. Initialization: Each node in the system is assigned a unique ID, and all nodes are aware of the existence of other nodes in the system.
2. Election Trigger: When a node wants to start an election, it sends an "ELECTION" message to all other nodes with higher IDs. This indicates that the initiating node wants to become the leader.
3. Response to Election Message: Nodes receiving the "ELECTION" message compare their IDs with the sender's ID. If their ID is higher, they send an "OK" message back to the initiating node, indicating that they are still active and have a higher priority.
4. Leader Election: If the initiating node doesn't receive an "OK" message from any node with a higher ID, it declares itself the leader since it has the highest ID among the active nodes.
5. Coordinator Message: After becoming the leader, the node sends a "COORDINATOR" message to all other nodes, informing them of its leadership status.
6. Handling Coordinator Messages: Nodes receiving the "COORDINATOR" message update their knowledge of the leader and perform any necessary actions based on the leader's instructions.

```java
// bully

import java.util.*;

public class bully {
```

```
  static boolean state[] = new boolean[1000];
  static int n;

  static void up(int process) {
    if (state[process]) {
      System.out.println("Process is already up");
    } else {
      state[process] = true;
      mess(process);
    }
  }

  public static void down(int process) {
    if (!state[process]) {
      System.out.println("Process is already down");
    } else {
      state[process] = false;
      System.out.println("Process down successfully");
    }
  }

  static void mess(int process) {
    if (state[process]) {
      System.out.println("Process" + process + " intiate election");
      for (int i = process + 1; i < n; i++) {
        System.out.println(
          "election message send from process " + process + " to
process " + i
        );
      }
      for (int i = n - 1; i >= process; i--) {
        if (state[i]) {
```

```
          System.out.println("Coordinator message send from " + i + "
to all");
          break;
        }
      }
    } else {
      System.out.println("Process" + process + " is down.");
    }
  }

  public static void main(String[] args) {
    int choice = 0, process = 0;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter Number of Process : ");
    n = sc.nextInt();
    for (int i = 0; i < n; i++) {
      state[i] = true;
    }
    System.out.println("There are " + n + "Processes are active : ");
    for (int i = 0; i < n; i++) System.out.print("p" + i + " ");
    System.out.println("\nProcess " + n + " is coordinator.");

    do {
      System.out.println("1. Up the process");
      System.out.println("2. Down the process");
      System.out.println("3. send message");
      System.out.print("Enter the choice : ");
      choice = sc.nextInt();

      switch (choice) {
        case 1:
          System.out.print("Bring Up Process : ");
          process = sc.nextInt();
          up(process);
```

```
          break;
        case 2:
          System.out.print("Bring Down Process : ");
          process = sc.nextInt();
          down(process);
          break;
        case 3:
          System.out.print("Message send by Process : ");
          process = sc.nextInt();
          mess(process);
          break;
      }
    } while (choice != 4);
  }
}
```

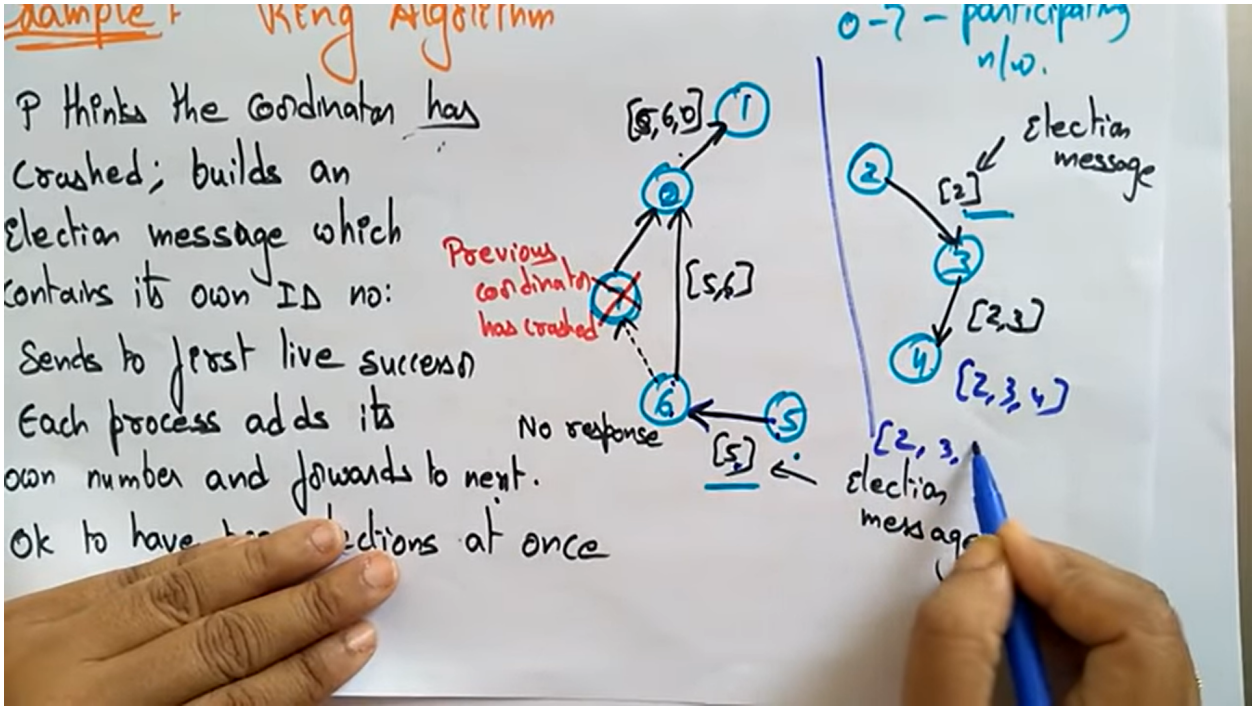
RING ALGORITHM :

This algorithm applies to systems organized as a ring(logically or physically). In this algorithm we assume that the link between the process are unidirectional and every process can message to the process on its right only. Data structure that this algorithm uses is **active list**, a list that has a priority number of all active processes in the system

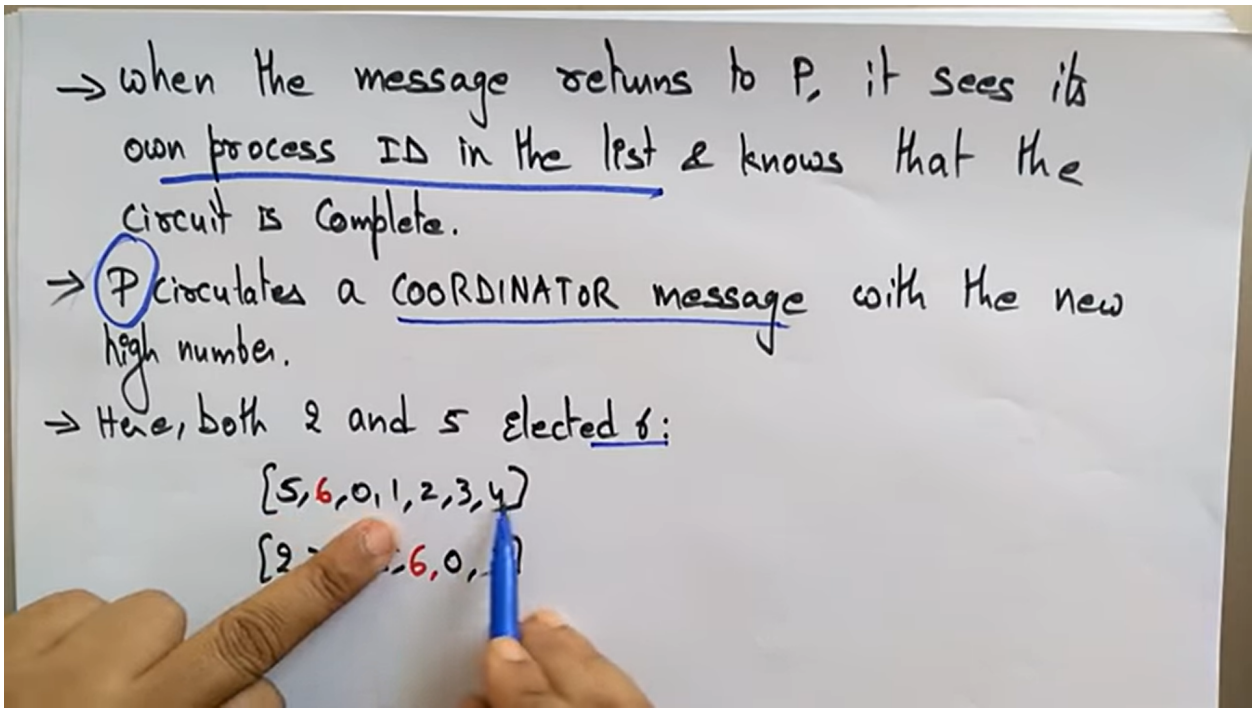
1. If process P1 detects a coordinator failure, it creates new active list which is empty initially. It sends election message to its neighbour on right and adds number 1 to its active list.

2. If process P2 receives message elect from processes on left, it responds in 3 ways:
    - (I) If message received does not contain 1 in active list then P1 adds 2 to its active list and forwards the message.
    - (II) If this is the first election message it has received or sent, P1 creates new active list with numbers 1 and 2. It then sends election message 1 followed by 2.
    - (III) If Process P1 receives its own election message 1 then active list for P1 now contains numbers of all the active processes in the system. Now Process P1 detects highest priority number from list and elects it as the new coordinator.

- If any node p thinks that the previous coordinator is crashed it has to build an election message that contains its own ID no.
- Send this to the first live successor
- Each process adds its own number and forwards the list to the next
- It is OK to have two elections at once
- When the message returns to P it checks for its own process id and knows that the circuit is completed



- P now circulates the coordinator message with the new highest number in the list



Here 6 gets elected now

What is the algorithm now ??

1. Declare state boolean array and n
2. In main function
3. Take the input for number of processes
4. Initialize state[0-n] to true
5. Initialize arr of n and arr[i] = -1
6. Print that there are n processes
7. Print n-1 process fails
8. State[n-1] = false;
9. Switch variable
10. Switch (choice) :

11.1 . Election

12. 2. Exit
13. Election main fir
14. Enter the initializing process
15. Process = sc.nextInt()
16. Cur = process
17. Do {}while(cur!=process);
18. Arr[j++] = cur;
19. Next = cur + 1 % n ;
20. while(!state[next])Next = (next + 1) %n;
21. Process cur sends message to next
22. Cur = next;
23. Find maximum of the array

24. Cur = process
25. Do a while loop
26. Process cur Pass Coordinator to maxi message to next
27. while(cur!=process)
28. Return
29. Break

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

public class ring {
    static boolean[] state = new boolean[1000];
    static int n;
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of Processes : ");
        n = sc.nextInt();
        for(int i = 0;i<n;i++){
            state[i] = true;
        }
        int[] arr = new int[n];
        for(int i = 0;i<n;i++){
            arr[i] = -1;
        }


        System.out.println("There are "+n + "Processses");
        for(int i = 0;i<n;i++){
            System.out.print("p"+i+"");
        }
        System.out.println("Process" + (n-1)+ "fails");


        state[n-1] = false;
```

```
        int choice = 0;

        do{
            System.out.println("1. Election");
            System.out.println("2. Exit");
            System.out.println("Enter your choice");

            choice = sc.nextInt();
            switch(choice){
                case 1 : {
                    System.out.println("Enter Process who is initializing the
election");
                    int process = sc.nextInt();
                    int cur = process , j = 0;
                    do {
                        arr[j++] = cur;
                        int next = (cur+1)%n;
                        while(!state[next]){
                            next =( next + 1 ) %n;
                        }
                        System.out.println("Process " + cur + "sends message to"+
next);
                        cur = next;

                    }while(cur!=process);


                    int maxi = -1;
                    for(int i = 0;i<n;i++) if(maxi<arr[i]) maxi = arr[i];
                    System.out.println("Process " + maxi + "Selected as
coordinator");
                    cur = process;

                    do{
                        int next = ( cur + 1 ) %n;
                        while(!state[next] ) next = (next + 1 ) % n;
                        System.out.println("Process " + cur + "Pass Coordinator ( " +
maxi + ") message to" + next);
                        cur = next;
                    }while(cur!=process);
```

```
                }
                case 2 : {
                    return ;
                }
                default : {
                    break;
                }
            }

        }while(true);
    }
}
```

Problem Statement 09 :

Create a simple web service and write any distributed application to consume the web service

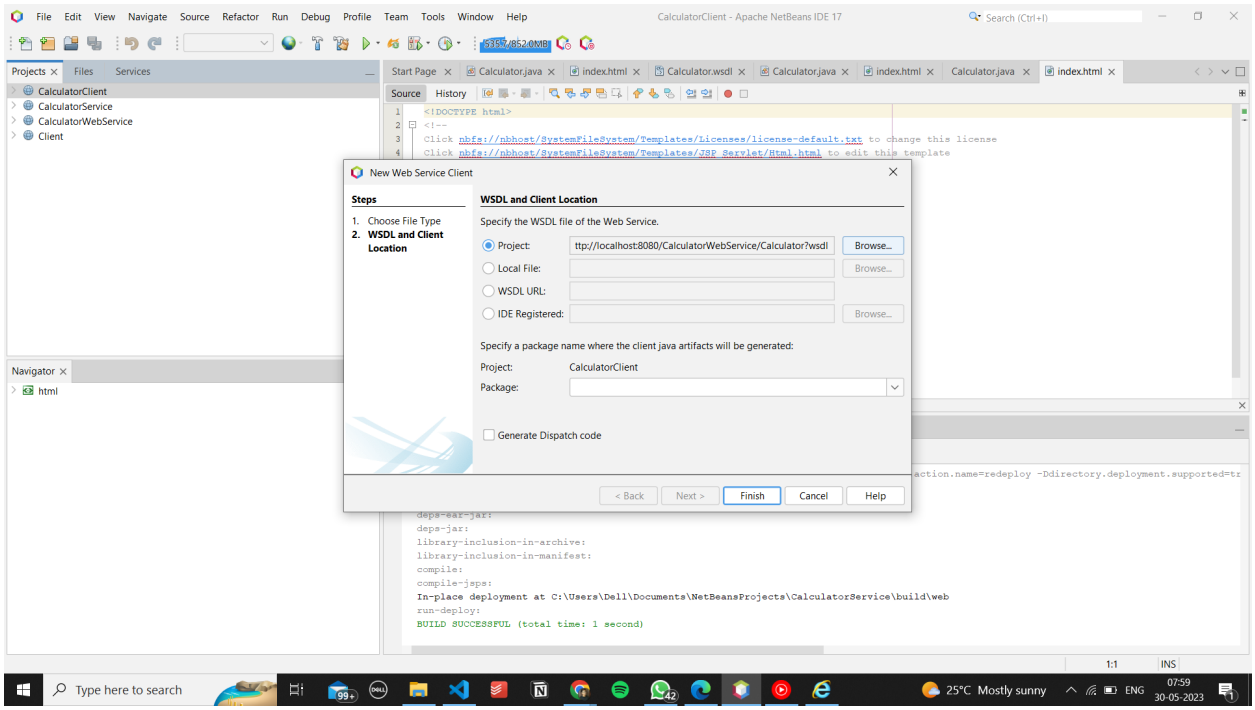
In Java, a web service is a software component or application that allows different systems to communicate and exchange data over the web using standard web protocols

It enables interoperability between various platforms and programming languages by providing a standardized way of exposing functionalities and accessing them remotely.

A web service in Java is typically implemented using the Java API for XML Web Services (JAX-WS) or Java API for RESTful Web Services (JAX-RS),

Steps :

1. Create web service
2. New Project
3. Java web → Web application
4. Project name
5. Right click on project create web service
6. Package com.unique
7. Delete hello world
8. Click on service add operation then select the parameter and two numbers int
9. Clean and build
10. Deploy
11. Test web service
12. Test Web service
13. Create a client similar to the new package
14. Create and new web service client
15. Pass the path of the web service



16. Web services references will contain the created method
17. Edit the index.html file
18. Name the action of the form calculator
19. Go to tools palette and select a form
20. Select input txt1
21. Select input txt2
22. Button submit
23. Now run the client
24. Go to source packages → new → servlet (name of the servlet should be same as the name of the action in the form
25. Add information to descriptor check krna hai
26. Now drag and drop the getNumber method onto the lower end of the servlet page

27. Now inside the processRequestMethod and inside try take input of the two number s

```
int num1 ,num2;
num1 = Integer.parseInt(request.getParameter("txt1"));
num2 = Integer.parseInt(request.getParameter("txt2"));
out.println("<!DOCTYPE html>");
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet calculator</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>" +getNumber(num1,num2)+ "</h1>");
```