Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA: 3.18)

Department of Computer Engineering

Academic Year 2022-2023

NAME	**:**	**Junaid Girkar**
SAP ID	**:**	**60004190057**
DIVISION	**:**	**A/A2**
BRANCH	**:**	**Computer Engineering**
A.Y.	**:**	**2022-23**
SEM	**:**	**VII**
SUB	**:**	**Distributed Computing Lab**

ACADEMIC YEAR – 2022-2023

List of Experiment

Sub- Distributed Computing Lab **Sem- VII**

1. Demonstrate Client / Server using RMI
2. Implement a program to demonstrate multithreading
3. Demonstration of IPC (Inter Process Communication)
4. Demonstration of Group Communication using Java
5. Implementation of Clock Synchronization Algorithm - Berkeley Algorithm
6. Implementation of Load Balancing Algorithm
7. Demonstration of Name Resolution Protocol
8. Implementation of Bully Election Algorithm
9. Implementation of Ring Election Algorithm
10. Demonstration of Deadlock Management
11. Demonstration of CORBA
12. Implementation of Hadoop Distributed File System

Prof. Aniket Kore

Prof. Pankaj Sonawane Dr. Meera Narvekar

Subject Incharges HoD, Computer Dept

EXPERIMENT - 1

Aim: Demonstrate Client / Server using RMI

Theory:

The RMI (Remote Method Invocation) is an API that provides a mechanism to create a distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects, stub and skeleton.

Understanding stub and skeleton
RMI uses stub and skeleton objects for communication with the remote object.

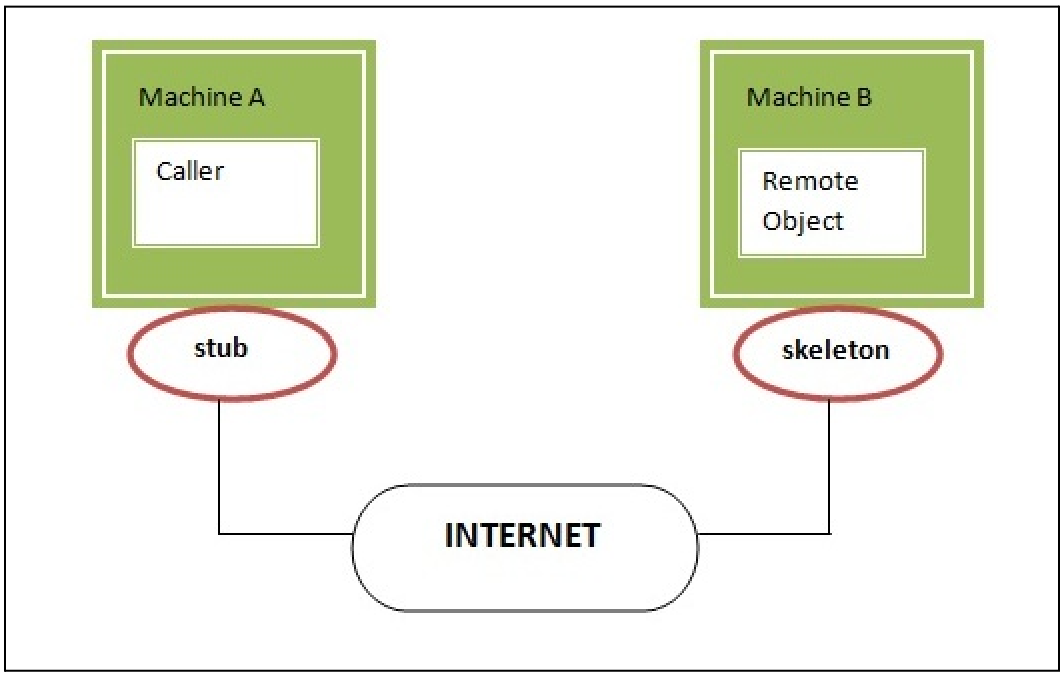
A remote object is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

1. **Stub**
   The stub is an object that acts as a gateway for the client side. All the outgoing requests are routed through it. It resides on the client side and represents the remote object. When the caller invokes a method on the stub object, it does the following tasks:

- It initiates a connection with a remote Virtual Machine (JVM),
- It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
- It waits for the result
- It reads (unmarshals) the return value or exception, and
- It finally returns the value to the caller.

2. **Skeleton**
   The skeleton is an object that acts as a gateway for the server-side object. All incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

- It reads the parameter for the remote method
- It invokes the method on the actual remote object, and
- It writes and transmits (marshals) the result to the caller.



Code:

```
//Server.java

import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server extends InterfaceImplementation {
  public Server() {}
  public static void main(String args[]) {
    try {
      // Instantiating the implementation class
      InterfaceImplementation obj = new InterfaceImplementation();

      // Exporting the object of implementation class
      // (here we are exporting the remote object to the stub)
        AdditionInterface stub = (AdditionInterface) UnicastRemoteObject.exportObject(obj,
0);

      // Binding the remote object (stub) in the registry
      Registry registry = LocateRegistry.getRegistry();

      registry.bind("AdditionInterface", stub);
      System.err.println("Server ready");
    } catch (Exception e) {
      System.err.println("Server exception: " + e.toString());
      e.printStackTrace();
    }
  }
}
```

```
//Client.java

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.*;

public class Client {
  private Client() {}
  public static void main(String[] args) {
    try {
      // Getting the registry
      Registry registry = LocateRegistry.getRegistry(null);
```

```
        // Looking up the registry for the remote object
        AdditionInterface stub = (AdditionInterface) registry.lookup("AdditionInterface");

       Scanner sc = new Scanner(System.in);
       System.out.print("Enter two numbers: ");
       int a = sc.nextInt();
       int b = sc.nextInt();
       sc.close();
        // Calling the remote method using the obtained object
        stub.addTwoNumbers(a, b);

        // System.out.println("Remote method invoked");
      } catch (Exception e) {
        System.err.println("Client exception: " + e.toString());
        e.printStackTrace();
      }
    }
}
```

Output:

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 2 -- RMI>start rmiregistry

C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 2 -- RMI>java Server
Server ready
The sum of 23 and 76 is: 99
```

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 2 -- RMIjava Client
Enter two numbers: 23 76
```

Conclusion:
Remote Method Invocation (RMI) is a form of Inter-Process Communication (IPC) that allows an abject to invoke methods on a remote object. It utilizes a combination of proxies, skeleton and dispatcher classes to perform remote processing without the application needing to perform networking functionality. In this experiment, we have demonstrated RMI using Java.

EXPERIMENT - 2

Aim: Implement a program to demonstrate multithreading

Theory:

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading are both used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory areas, so it saves memory, and context-switching between the threads takes less time than the process.

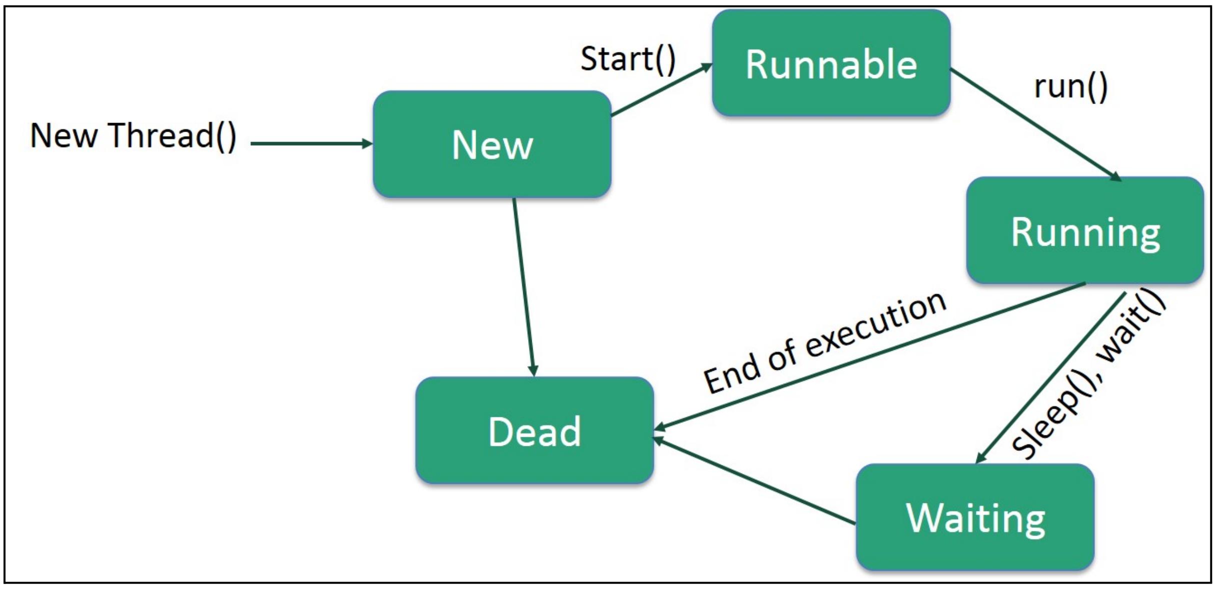
Java Multithreading is primarily used in games, animation, etc.

Life Cycle of a Thread

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.

Java Thread

Following are the stages of the life cycle –



- ➔ **New** – A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.

- ➔ **Runnable** – After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.

- ➔ **Waiting** – Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

- ➔ **Timed Waiting** – A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.

- ➔ **Terminated (Dead)** – A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Threads can be created by using two mechanisms:

- Extending the Thread class
- Implementing the Runnable Interface

Code:

```
// Tables.java
public class Tables {

  public static void main(String[] args) {
    Five f = new Five();
    Seven s = new Seven();
    Thirteen t = new Thirteen();
    f.start();
    s.start();
    t.start();
  }
}

class Seven extends Thread {

  public void run() {
    long t1 = System.currentTimeMillis();
    for (int i = 1; i <= 10; i++) {
      System.out.println("7 x " + i + " = " + (7 * i));
      try {
        Thread.sleep(1000);
      } catch (Exception e) {}
    }
    long t2 = System.currentTimeMillis();
    System.out.println("Time Taken by table of 7: " + (t2 - t1) + " ms ");
  }
}

class Five extends Thread {

  public void run() {
    long t1 = System.currentTimeMillis();
    for (int i = 1; i <= 10; i++) {
      System.out.println("5 x " + i + " = " + (5 * i));
      try {
        Thread.sleep(1000);
      } catch (Exception e) {}
    }
    long t2 = System.currentTimeMillis();
    System.out.println("Time Taken by table of 5: " + (t2 - t1) + " ms ");
  }
}

class Thirteen extends Thread {

  public void run() {
    long t1 = System.currentTimeMillis();
    for (int i = 1; i <= 10; i++) {
      System.out.println("13 x " + i + " = " + (13 * i));
      try {
        Thread.sleep(1000);
      } catch (Exception e) {}
    }
    long t2 = System.currentTimeMillis();
    System.out.println("Time Taken by table of 13: " + (t2 - t1) + " ms ");
  }
}
```

Output:

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 3 -- Multithreading>java Tables
7 x 1 = 7
13 x 1 = 13
5 x 1 = 5
5 x 2 = 10
```

```
7 x 2 = 14
13 x 2 = 26
5 x 3 = 15
13 x 3 = 39
7 x 3 = 21
5 x 4 = 20
13 x 4 = 52
7 x 4 = 28
5 x 5 = 25
7 x 5 = 35
13 x 5 = 65
7 x 6 = 42
13 x 6 = 78
5 x 6 = 30
13 x 7 = 91
5 x 7 = 35
7 x 7 = 49
13 x 8 = 104
5 x 8 = 40
7 x 8 = 56
5 x 9 = 45
13 x 9 = 117
7 x 9 = 63
13 x 10 = 130
5 x 10 = 50
7 x 10 = 70
Time Taken by table of 7: 10114 ms
Time Taken by table of 13: 10114 ms
Time Taken by table of 5: 10115 ms
```

Conclusion:
Multithreaded applications utilize a collection of threads representing a process to execute concurrently with separate context and scheduling concerns. This approach increases the parallelism of the system and increases productivity by a linear factor. In this experiment, we have demonstrated multithreading using a Java program.

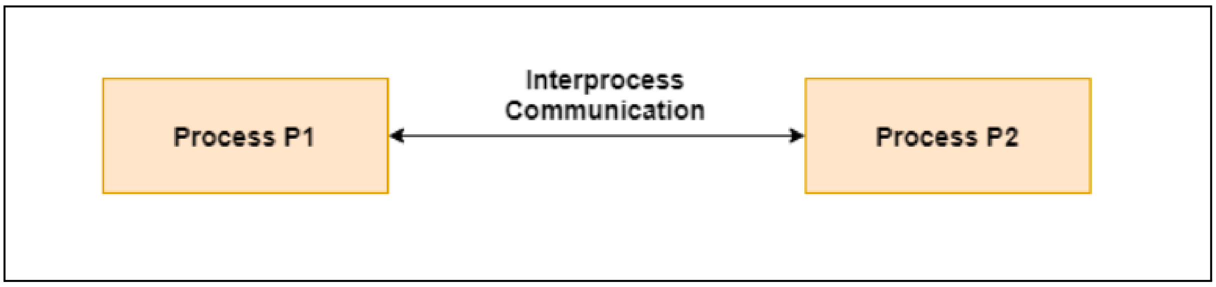
EXPERIMENT - 3

Aim: Demonstration of IPC (Inter Process Communication)

Theory:

Interprocess communication is the mechanism provided by the operating system that allows processes to communicate with each other. This communication could involve a process letting another process know that some event has occurred or the transferring of data from one process to another.

A diagram that illustrates interprocess communication is as follows –



Synchronization in Interprocess Communication

Synchronization is a necessary part of interprocess communication. It is either provided by the interprocess control mechanism or handled by the communicating processes. Some of the methods to provide synchronization are as follows –

➔ **Semaphore**
A semaphore is a variable that controls the access to a common resource by multiple processes. The two types of semaphores are binary semaphores and counting semaphores.

➔ **Mutual Exclusion**
Mutual exclusion requires that only one process thread can enter the critical section at a time. This is useful for synchronization and also prevents race conditions.

➔ **Barrier**
A barrier does not allow individual processes to proceed until all the processes reach it. Many parallel languages and collective routines impose barriers.

➔ **Spinlock**
This is a type of lock. The processes trying to acquire this lock wait in a loop while checking if the lock is available or not. This is known as busy waiting because the process is not doing any useful operation even though it is active.

Approaches to Interprocess Communication

The different approaches to implement interprocess communication are given as follows –

Pipe

A pipe is a data channel that is unidirectional. Two pipes can be used to create a two-way data channel between two processes. This uses standard input and output methods. Pipes are used in all POSIX systems as well as Windows operating systems.

Socket

The socket is the endpoint for sending or receiving data in a network. This is true for data sent between processes on the same computer or data sent between different computers on the same network. Most of the operating systems use sockets for interprocess communication.

File

A file is a data record that may be stored on a disk or acquired on demand by a file server. Multiple processes can access a file as required. All operating systems use files for data storage.

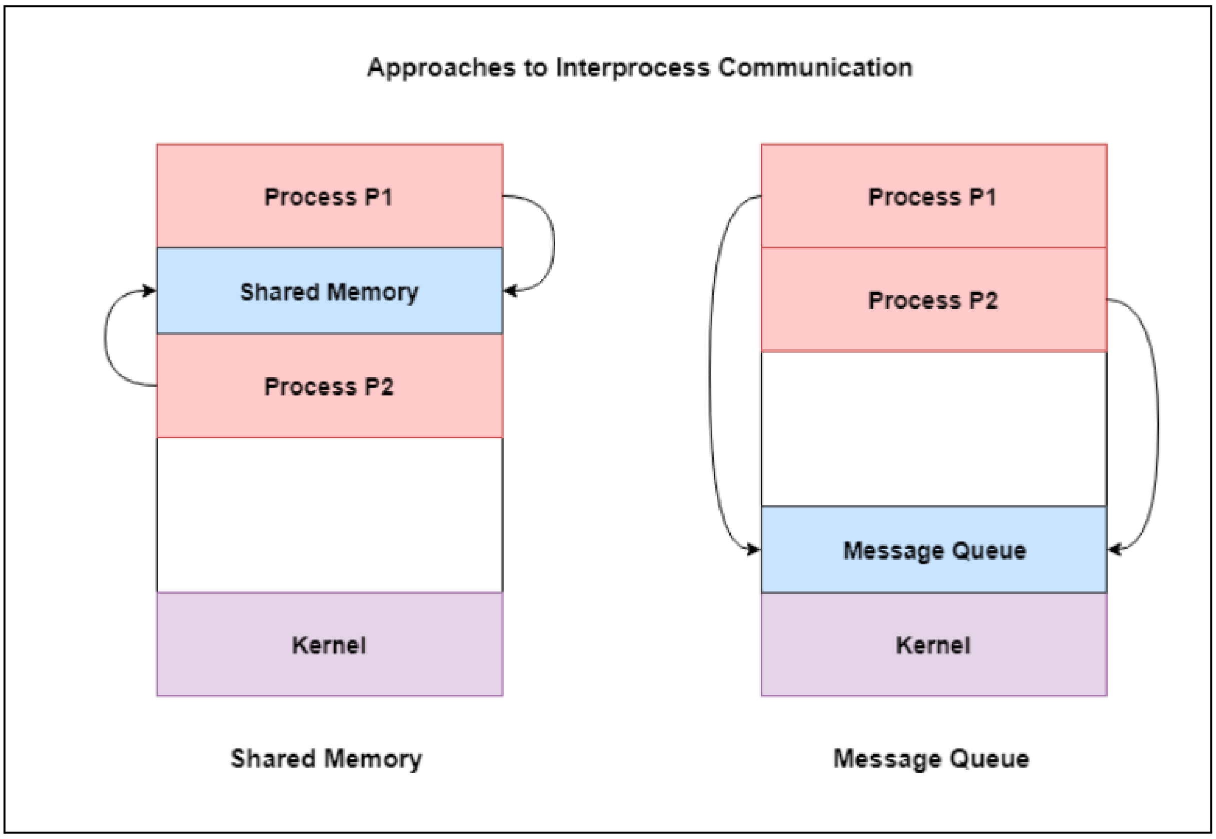
Signal

Signals are useful in interprocess communication in a limited way. They are system messages that are sent from one process to another. Normally, signals are not used to transfer data but are used for remote commands between processes.

Shared Memory
Shared memory is the memory that can be simultaneously accessed by multiple processes. This is done so that the processes can communicate with each other. All POSIX systems, as well as Windows operating systems use shared memory.

Message Queue
Multiple processes can read and write data to the message queue without being connected to each other. Messages are stored in the queue until their recipient retrieves them. Message queues are quite useful for interprocess communication and are used by most operating systems.

A diagram that demonstrates message queue and shared memory methods of interprocess communication is as follows –



Code:

```
//IPCServer.java

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

public class IPCServer {
   public static void main(String args[]) {
      System.out.println("\n **** INTERPROCESS COMMUNICATION ****\n");
      System.out.println("\n *** SERVER PROCESS STARTED ***\n");
        System.out.println("\n * SERVER IS READY AND WAITING TO RECEIVE DATA
FROM CLIENT PROCESS ON PORT " + 1200);
      try {
         ServerSocket ss = new ServerSocket(1200); // 1200 is port number
         Socket clientSocket = ss.accept();
                     System.out.println("\n * Client is connected with IP address " +
clientSocket.getInetAddress()
                + " and port Number " + clientSocket.getPort());
                                             DataOutputStream dos = new
DataOutputStream(clientSocket.getOutputStream());
                                               DataInputStream dis = new
DataInputStream(clientSocket.getInputStream());
         int a = dis.readInt();
         System.out.println("\n SERVER RECEIVED");
         System.out.println("\n Number 1 ====>" + a);
         int b = dis.readInt();
         System.out.println("\n Number 2 ====>" + b);
         int c = a + b;
         dos.writeInt(c);
         System.out.println(
                    "\n SERVER PROCESS HAS EXECUTED REQUESTED PROCESS AND
SENT RESULT " + c + " TO THE CLIENT \n");
         clientSocket.close();
         System.out.println("\n SERVER PROCESS EXITING........");
         ss.close();
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }
}
```

```
// IPCClient.java

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

public class IPCClient {
    public static void main(String args[]) {
        try {
            Socket s = new Socket("localhost", 1200);
            DataOutputStream dos = new DataOutputStream(s.getOutputStream());
            DataInputStream dis = new DataInputStream(s.getInputStream());
            InputStreamReader isr = new InputStreamReader(System.in);
                    System.out.println(" \n \t************* CLIENT PROCESS STARTED
********************* ");
            System.out.println(
                    "\n ********* PLEASE ENTER THE VALUES OF Number 1 AND Number 2
TO PASS THEM TO SERVER PROCESS******** \n");
            BufferedReader br = new BufferedReader(isr);
            int a = Integer.parseInt(br.readLine());
            System.out.println("Number 1 ====>" + a);
            dos.writeInt(+a);
            int b = Integer.parseInt(br.readLine());
            System.out.println("Number 2 ====>" + b);
            dos.writeInt(+b);
            int result = dis.readInt();
                System.out.println("\n............CLIENT PROCESS HAS RECEIVED RESULT
FROM SERVER..............\n");
```

```
            System.out.println("\n THE ADDITION OF " + a + " AND " + b + " IS " + result);
            s.close();
        } catch (Exception e) {
            System.out.println("Exception is " + e);
        }
    }
}
```

Output:

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM 7\Distributed Computing\Practical X – IPCjava IPCServer

 **** INTERPROCESS COMMUNICATION ****


 *** SERVER PROCESS STARTED ***


 * SERVER IS READY AND WAITING TO RECEIVE DATA FROM CLIENT PROCESS ON PORT 1200

 * Client is connected with IP address /127.0.0.1 and port Number 62153

 SERVER RECEIVED

 Number 1 ====>23

 Number 2 ====>65

 SERVER PROCESS HAS EXECUTED REQUESTED PROCESS AND SENT RESULT 88 TO THE CLIENT
SERVER PROCESS EXITING.........
```

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical X – IPCjava IPCClient

     ************** CLIENT PROCESS STARTED **********************

 ********* PLEASE ENTER THE VALUES OF Number 1 AND Number 2 TO PASS
THEM TO SERVER PROCESS********

23
Number 1 ====>23
65
Number 2 ====>65

............. CLIENT PROCESS HAS RECEIVED RESULT FROM SERVER..............


 THE ADDITION OF 23 AND 65 IS 88
```

Conclusion:
Inter-Process Communication (IPC) is a message-oriented communication mechanism that allows various processes or tasks to transfer data and variables between each other. It allows a single program to be split and distributed among various nodes and have the state data flow between them as in a centralized processing environment. In this experiment, we have demonstrated IPC using Java.

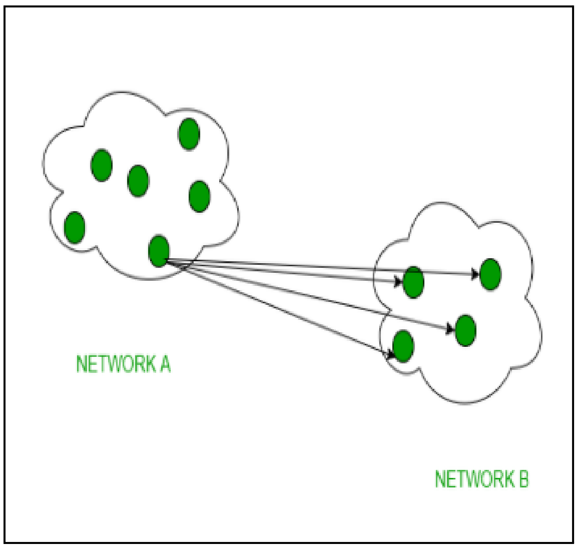
EXPERIMENT - 4

Aim: Demonstration of Group Communication using Java

Theory:
Broadcast messaging is a collection of techniques that people can use to deliver information to lots of recipients at once. For example, during a weather emergency, law enforcement officials can notify people of life-threatening conditions. Doctors can let patients know when the season's flu shots have arrived. Staff at an electronics store can inform customers that a highly anticipated game is available for pre-order. Sometimes, this information is convenient; other times, it's critical.

The goal of broadcast messaging is to reach as many people as possible as quickly as possible.

Broadcast transfer (one-to-all) techniques and can be classified into two types : Limited Broadcasting, Direct Broadcasting. In broadcasting mode, transmission happens from one host to all the other hosts connected on the LAN. The devices such as bridge uses this. The protocol like ARP implement this, in order to know MAC address for the corresponding IP address of the host machine. ARP does ip address to mac address translation. RARP does the reverse.



Code:

```
// Server.java

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Vector;

public class Server {

  private static Vector<PrintWriter> writers = new Vector<PrintWriter>();

  public static void main(String[] args) throws Exception {
    ServerSocket listener = new ServerSocket(9001);
    System.out.println("The server is running at port 9001.");
    while (true)
      new Handler(listener.accept()).start();
  }

  private static class Handler extends Thread {

    private Socket socket;

    public Handler(Socket socket) {
      this.socket = socket;
    }

    public void run() {
      try {
                                    BufferedReader    in    =    new    BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        out.println("SUBMITNAME");
        String name = in.readLine();
        System.out.println(name + " joined");
        writers.add(out);
        while (true) {
          String input = in.readLine();
          for (PrintWriter writer : writers)
            writer.println(
                "MESSAGE " + name + ": " + input);
        }
      } catch (Exception e) {
        System.err.println(e);
      }
    }
  }
}
```

```
// Master.java

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class Master {

  public static void main(String[] args) throws Exception {
    Scanner sc = new Scanner(System.in);
    Socket socket = new Socket("localhost", 9001);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));
    PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
    System.out.print("Enter your name: ");
    String name = sc.nextLine();
    while (true) {
      String line = in.readLine();
      if (line.startsWith("SUBMITNAME"))
        out.println(name);
      else if (line.startsWith("MESSAGE"))
        System.out.println(line.substring(8));
      if (name.startsWith("master")) {
        System.out.print("Enter a message: ");
        out.println(sc.nextLine());
      }
    }
  }
}
```

```
// Slave1.java

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class slave1 {

  public static void main(String[] args) throws Exception {
    Scanner sc = new Scanner(System.in);
    Socket socket = new Socket("localhost", 9001);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));
    PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
    System.out.print("Enter your name: ");
    String name = sc.nextLine();
    while (true) {
      String line = in.readLine();
      if (line.startsWith("SUBMITNAME"))
        out.println(name);
      else if (line.startsWith("MESSAGE"))
```

```
          System.out.println(line.substring(8));
        if (name.startsWith("master")) {
          System.out.print("Enter a message: ");
          out.println(sc.nextLine());
        }
      }
    }
}
```

```
// Slave2.java

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class slave2 {

  public static void main(String[] args) throws Exception {
    Scanner sc = new Scanner(System.in);
    Socket socket = new Socket("localhost", 9001);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));
    PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
    System.out.print("Enter your name: ");
    String name = sc.nextLine();
    while (true) {
      String line = in.readLine();
      if (line.startsWith("SUBMITNAME"))
        out.println(name);
      else if (line.startsWith("MESSAGE"))
        System.out.println(line.substring(8));
      if (name.startsWith("master")) {
        System.out.print("Enter a message: ");
        out.println(sc.nextLine());
      }
    }
  }
}
```

Output:

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 3 -- Broadcasting Msg>java Server
The server is running at port 9001.
master joined
junaid1 joined
junaid2 joined
```

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 3 -- Broadcasting Msg>java Master
Enter your name: master
Enter a message: Hello World
master: Hello World
Enter a message: This is the Second Message
master: This is the Second Message
Enter a message:
```

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 3 -- Broadcasting Msg>java slave1
Enter your name: junaid1
master: Hello World
master: This is the Second Message
```

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
```

```
7\Distributed Computing\Practical 3 -- Broadcasting Msg>java slave2
Enter your name: junaid2
master: Hello World
master: This is the Second Message
```

Conclusion:
Group communication allows a diverse set interaction between processes by allowing a single process to communicate with multiple processes simultaneously and vice versa. It foregoes point-to-point communication and groups target nodes as an entity to reduce communication overhead. It includes unicast, multicast, and broadcast communication. In this experiment, we have demonstrated group communication using Java.

EXPERIMENT - 5

Aim: Implementation of Clock Synchronization Algorithm - Berkeley Algorithm

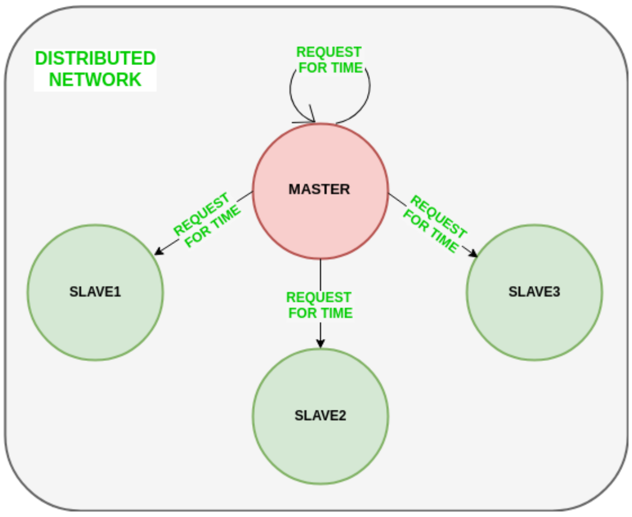
Theory:

Berkeley's Algorithm is a clock synchronization technique used in distributed systems. The algorithm assumes that each machine node in the network either doesn't have an accurate time source or doesn't possess a UTC server.

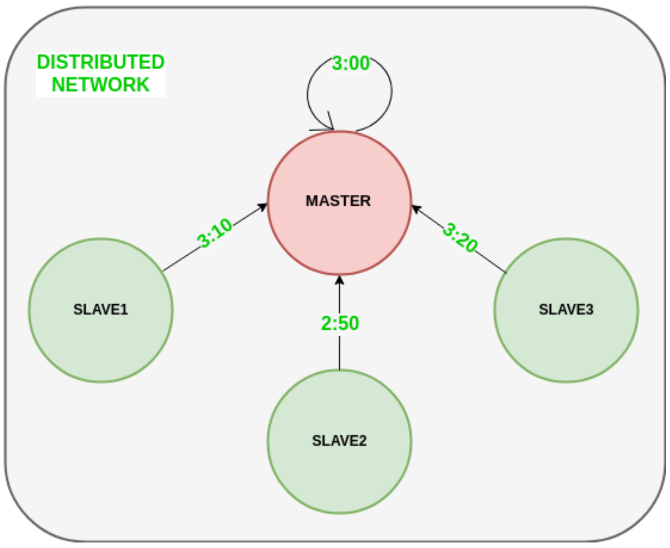
Algorithm

1. An individual node is chosen as the master node from a pool node in the network. This node is the main node in the network which acts as a master and the rest of the nodes act as slaves. The master node is chosen using an election process/leader election algorithm.
2. Master node periodically pings slaves nodes and fetches clock time at them using Cristian's algorithm.

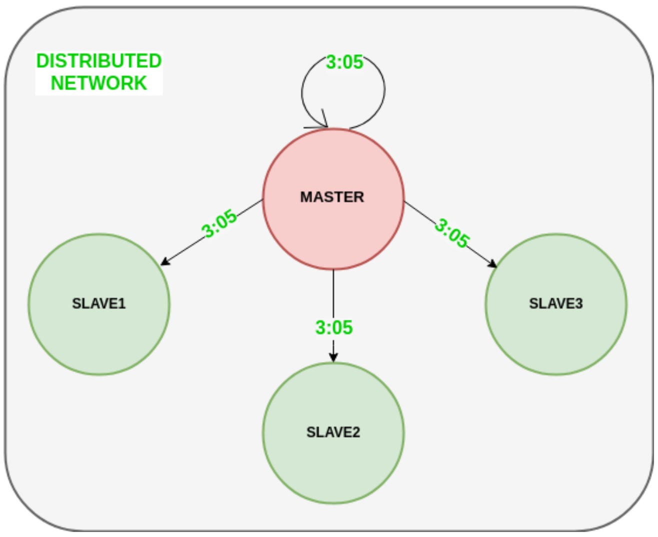
   The diagram below illustrates how the master sends requests to slave nodes.



The diagram below illustrates how slave nodes send back time given by their system clock.



3. Master node calculates the average time difference between all the clock times received and the clock time given by the master's system clock itself. This average time difference is added to the current time at the master's system clock and broadcasted over the network.



Code:

```
// Berkeley.java

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

public class Berkeley {
    float diff(int h, int m, int s, int nh, int nm, int ns) {
        int dh = h - nh;
        int dm = m - nm;
        int ds = s - ns;
        int diff = (dh * 60 * 60) + (dm * 60) + ds;
        return diff;
    }

    float average(float diff[], int n) {
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += diff[i];
        }
        float average = (float) sum / (n + 1);
        System.out.println("The average of all time differences is " + average);
        return average;
```

```
}

void sync(float diff[], int n, int h, int m, int s, int nh[], int nm[], int ns[], float average) {
    for (int i = 0; i < n; i++) {
        diff[i] += average;
        int dh = (int) diff[i] / (60 * 60);
        diff[i] %= (60 * 60);
        int dm = (int) diff[i] / 60;
        diff[i] %= 60;
        int ds = (int) diff[i];
        nh[i] += dh;
        if (nh[i] > 23) {
            nh[i] %= 24;
        }
        nm[i] += dm;
        if (nm[i] > 59) {
            nh[i]++;
            nm[i] %= 60;
        }
        ns[i] += ds;
        if (ns[i] > 59) {
            nm[i]++;
            ns[i] %= 60;
        }
        if (ns[i] < 0) {
            nm[i]--;
            ns[i] += 60;
        }
    }
    h += (int) (average / (60 * 60));
    if (h > 23) {
        h %= 24;
    }
    m += (int) (average / (60 * 60 * 60));
    if (m > 59) {
        h++;
        m %= 60;
    }
```

```
        s += (int) (average % (60 * 60 * 60));
        if (s > 59) {
            m++;
            s %= 60;
        }
        if (s < 0) {
            m--;
            s += 60;
        }
        System.out.println("The synchronized clocks are:\nTime Server ---> " + h + " : " + m + " :
" + s);
        for (int i = 0; i < n; i++) {
            System.out.println("Node " + (i + 1) + " ---> " + nh[i] + " : " + nm[i] + " : " + ns[i]);
        }
    }

    public static void main(String[] args) throws IOException {
        Berkley b = new Berkley();
        Date date = new Date();
        BufferedReader obj = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter number of nodes:");
        int n = Integer.parseInt(obj.readLine());
        int h = date.getHours();
        int m = date.getMinutes();
        int s = date.getSeconds();
        int nh[] = new int[n];
        int nm[] = new int[n];
        int ns[] = new int[n];
        for (int i = 0; i < n; i++) {
            System.out.println("Enter time for node " + (i + 1) + "\n Hours:");
            nh[i] = Integer.parseInt(obj.readLine());
            System.out.println("Minutes:");
            nm[i] = Integer.parseInt(obj.readLine());
            System.out.println("Seconds:");
            ns[i] = Integer.parseInt(obj.readLine());
        }
        for (int i = 0; i < n; i++) {
            System.out.println("Time Server sent time " + h + " : " + m + " : " + s + " to node " + (i +
1));
        }
        float diff[] = new float[n];
        for (int i = 0; i < n; i++) {
            diff[i] = b.diff(h, m, s, nh[i], nm[i], ns[i]);
               System.out.println("Node " + (i + 1) + " sent time difference of " + (int) diff[i] + " to
Time Server.");
        }
        float average = b.average(diff, n);
        b.sync(diff, n, h, m, s, nh, nm, ns, average);
    }
}
```

Output:

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical X2 -- Berkeley Clock Sync>java Berkeley
Enter number of nodes:
4
Enter time for node 1
 Hours:
3
Minutes:
12
Seconds:
2
Enter time for node 2
 Hours:
5
Minutes:
1
Seconds:
2
Enter time for node 3
 Hours:
1
Minutes:
```

```
3
Seconds:
4
Enter time for node 4
 Hours:
8
Minutes:
3
Seconds:
2
Time Server sent time 17 : 29 : 55 to node 1
Time Server sent time 17 : 29 : 55 to node 2
Time Server sent time 17 : 29 : 55 to node 3
Time Server sent time 17 : 29 : 55 to node 4
Node 1 sent time difference of 51473 to Time Server.
Node 2 sent time difference of 44933 to Time Server.
Node 3 sent time difference of 59211 to Time Server.
Node 4 sent time difference of 34013 to Time Server.
The average of all time differences is 37926.0
The synchronized clocks are:
Time Server ---> 3 : 30 : 1
Node 1 ---> 4 : 2 : 1
Node 2 ---> 4 : 2 : 1
Node 3 ---> 4 : 2 : 1
Node 4 ---> 4 : 2 : 1
```

Conclusion:

Berkley's algorithm is a clock synchronization method that allows multiple processes to agree on a certain time within a local context. It starts on the assumption that no clock is perfect, it, therefore, tries to average out the times of the various member nodes to settle a local clock for the network. It utilizes time difference deltas between the nodes and the round trip time (RTT) between the nodes to generate accurate measurements. In this experiment, we have implemented Berkley's algorithm using Java.

EXPERIMENT - 6

Aim: Implementation of Load Balancing Algorithm

Theory:

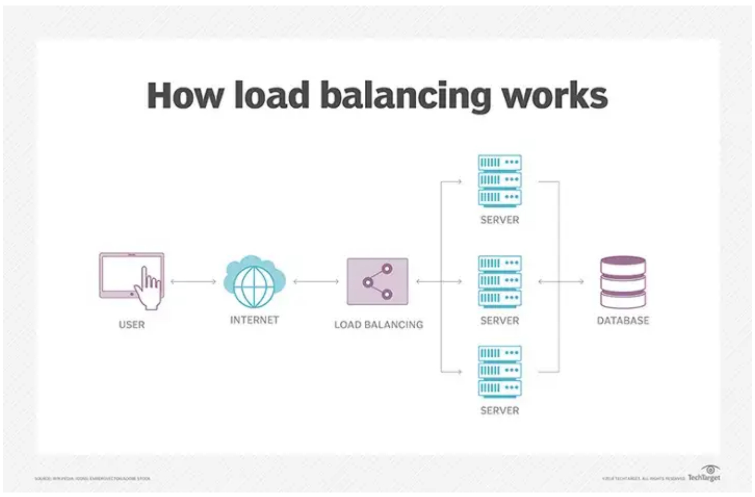
Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a server farm or server pool.

Modern high-traffic websites must serve hundreds of thousands, if not millions, of concurrent requests from users or clients and return the correct text, images, video, or application data, all in a fast and reliable manner. Modern computing best practice generally requires adding more servers to scale to meet these high volumes to cost-effectively scale.

A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers, capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. The load balancer redirects traffic to the remaining online servers if a single server goes down. When a new server is added to the server group, the load balancer automatically sends requests to it.

In this manner, a load balancer performs the following functions:

- ➔ Distributes client requests or network load efficiently across multiple servers
- ➔ Ensures high availability and reliability by sending requests only to servers that are online
- ➔ Provides the flexibility to add or subtract servers as demand dictates



Code:

```
// LoadBalancer.java

import java.util.Scanner;

class LoadBalancer {

 static void printLoad(int servers, int Processes) {
  int each = Processes / servers;
  int extra = Processes % servers;
  int total = 0;

  for (int i = 0; i < servers; i++) {
   if (extra-- > 0)
    total = each + 1;
   else
    total = each;
   System.out.println(
     "Server " + (char) ('A' + i) + " has " + total + " Processes");
  }
 }

 public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
  System.out.print("Enter the number of servers and Processes: ");
```

```
    int servers = sc.nextInt();
    int Processes = sc.nextInt();
    while (true) {
      printLoad(servers, Processes);
      System.out.print(
          "1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: ");
      switch (sc.nextInt()) {
        case 1:
          System.out.print("How many more servers?: ");
          servers += sc.nextInt();
          break;
        case 2:
          System.out.print("How many servers to remove?: ");
          servers -= sc.nextInt();
          break;
        case 3:
          System.out.print("How many more Processes?: ");
          Processes += sc.nextInt();
          break;
        case 4:
          System.out.print("How many Processes to remove?: ");
          Processes -= sc.nextInt();
          break;
        case 5:
          return;
      }
    }
  }
}
```

Output:

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 4 - Load Balancing>java LoadBalancer
Enter the number of servers and Processes: 3 8
```

```
Server A has 3 Processes
Server B has 3 Processes
Server C has 2 Processes

1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 1
How many more servers?: 2
Server A has 2 Processes
Server B has 2 Processes
Server C has 2 Processes
Server D has 1 Processes
Server E has 1 Processes

1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 2
How many servers to remove?: 1
Server A has 2 Processes
Server B has 2 Processes
Server C has 2 Processes
Server D has 2 Processes

1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 3
How many more Processes?: 5
Server A has 4 Processes
Server B has 3 Processes
Server C has 3 Processes
Server D has 3 Processes

1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 4
How many Processes to remove?: 2
Server A has 3 Processes
Server B has 3 Processes
Server C has 3 Processes
Server D has 2 Processes

1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 5
```

Conclusion:

Load balancing is a process management approach that aims to distribute resources among the available resources equally in a distributed network. Load balancing works by monitoring the load on each node and trying to equalize the utilization within a certain margin. This involves the implementation of certain policies including location estimation, thresholding, process priority, and migration policy among others.

EXPERIMENT - 7

Aim: Demonstration of Name Resolution Protocol

Theory:
In computer systems, name resolution refers to the retrieval of the underlying numeric values corresponding to computer hostnames, account user names, group names, and other named entities.

Computer operating systems commonly employ multiple key/value lists that associate easily remembered names with the integer numbers used to identify users, groups, other computers, hardware devices, and other entities. In that context, name resolution refers to the retrieval of numeric values given the associated names, while Reverse name resolution refers to the opposite process of finding the name(s) associated with specified numeric values:

Host identities
Hosts on a TCP/IP network have multiple identities. Network devices use these identities to deliver data to the correct hosts.

The three identities are the following:

- ➔ **Media access control (MAC) address**. The network interface card (NIC) has a MAC address encoded on its firmware.
- ➔ **IP address**. The NIC also has a logical IP address assigned to it.
- ➔ **Hostname**. The system has a human-friendly hostname set during the OS installation.

These identities provide a means of finding a given node on a network or network segment.

Code:

```
// NameResolution.java
```

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

public class NameResolution {

  public static void main(String args[]) throws IOException {
    while (true) {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      System.out.println(
        "\n Enter the website url (like google.com) to Resolve its Name to Address:");
      String name = br.readLine();
      try {
        InetAddress ip = InetAddress.getByName(name);
        System.out.println("\nIP Address: " + ip.getHostAddress());
      } catch (UnknownHostException e) {
        System.out.println("\n\n No such Host is present...");
        System.out.println("\n Try Again...");
      }
    }
  }
}
```

Output:

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 5 -- Name Resolution>java NameResolution

 Enter the website url (like google.com) to Resolve its Name to Address:
www.google.com

IP Address: 142.250.183.100

 Enter the website url (like google.com) to Resolve its Name to Address:
www.amazon.com
```

```
IP Address: 162.219.225.118

 Enter the website url (like google.com) to Resolve its Name to Address:
www.amazon.in

IP Address: 18.66.54.214

 Enter the website url (like google.com) to Resolve its Name to Address:
www.djsce.ac.in

IP Address: 148.251.191.4

 Enter the website url (like google.com) to Resolve its Name to Address:
```

Conclusion:
Name Resolution is a key concept of distributed communication. It enables the nodes on a network to identify dynamic addresses for other nodes based on a static key value. It thus allows flexibility and change in the network while still allowing nodes to operate as is without a change of configuration. This mechanism underlies many basic networking protocols including the Domain Naming System (DNS) and Address Resolution Protocol (ARP).

EXPERIMENT - 8

Aim: Implementation of Bully Election Algorithm

Theory:

Distributed Algorithm is an algorithm that runs on a distributed system. A distributed system is a collection of independent computers that do not share their memory. Each processor has its own memory, and they communicate via communication networks. Communication in networks is implemented in a process on one machine communicating with a process on another machine. Many algorithms used in the distributed system require a coordinator that performs functions needed by other processes in the system.

Election algorithms are designed to choose a coordinator.

Election Algorithms: Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes due to some reason, then a new coordinator is elected on another processor. The election algorithm basically determines where a new copy of the coordinator should be restarted. The election algorithm assumes that every active process in the system has a unique priority number. The process with the highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects the active process which has the highest priority number. Then this number is sent to every active process in the distributed system.

The Bully Algorithm – This algorithm applies to a system where every process can send a message to every other process in the system. Algorithm – Suppose process P sends a message to the coordinator.

1. If the coordinator does not respond to it within a time interval T, then it is assumed that the coordinator has failed.
2. Now process P sends an election message to every process with high priority number.

3. It waits for responses. If no one responds for time interval T then process P elects itself as a coordinator.
4. Then it sends a message to all lower priority number processes that it is elected as their new coordinator.
5. However, if an answer is received within time T from any other process Q,
    a. Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
    b. If Q doesn't responds within time interval T' then it is assumed to have failed and algorithm is restarted.

Code:

```
// BullyAlgo.java

import java.io.*;

class BullyAlgo {

  int cood, ch, crash;
  int prc[];

  public void election(int n) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("\nThe Coordinator Has Crashed!");
    int flag = 1;
    while (flag == 1) {
      crash = 0;
      for (int i1 = 0; i1 < n; i1++) if (prc[i1] == 0) crash++;
      if (crash == n) {
        System.out.println("\n*** All Processes Are Crashed ***");
        break;
      } else {
        System.out.println("\nEnter The Intiator");
        int init = Integer.parseInt(br.readLine());
        if ((init < 1) || (init > n) || (prc[init - 1] == 0)) {
          System.out.println("\nInvalid Initiator");
          continue;
        }
        for (int i1 = init - 1; i1 < n; i1++) System.out.println(
          "Process " + (i1 + 1) + " Called For Election"
        );
        System.out.println("");
        for (int i1 = init - 1; i1 < n; i1++) {
          if (prc[i1] == 0) {
            System.out.println("Process " + (i1 + 1) + " Is Dead");
          } else System.out.println("Process " + (i1 + 1) + " Is In");
        }
        for (int i1 = n - 1; i1 >= 0; i1--) if (prc[i1] == 1) {
          cood = (i1 + 1);
          System.out.println("\n*** New Coordinator Is " + (cood) + " ***");
          flag = 0;
          break;
        }
      }
    }
  }

  public void Bully() throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Enter The Number Of Processes: ");
    int n = Integer.parseInt(br.readLine());
    prc = new int[n];
    crash = 0;
    for (int i = 0; i < n; i++) prc[i] = 1;
    cood = n;
    do {
      System.out.println("\n\t1. Crash A Process");
      System.out.println("\t2. Recover A Process");
      System.out.println("\t3. Display New Cordinator");
      System.out.println("\t4. Exit");
      ch = Integer.parseInt(br.readLine());
      switch (ch) {
        case 1:
          System.out.println("\nEnter A Process To Crash");
          int cp = Integer.parseInt(br.readLine());
          if ((cp > n) || (cp < 1)) {
            System.out.println("Invaid Process! Enter A Valid Process");
          } else if ((prc[cp - 1] == 1) && (cood != cp)) {
            prc[cp - 1] = 0;
            System.out.println("\nProcess " + cp + " Has Been Crashed");
          } else if ((prc[cp - 1] == 1) && (cood == cp)) {
            prc[cp - 1] = 0;
            election(n);
          } else System.out.println("\nProcess " + cp + " Is Already Crashed");
          break;
        case 2:
          System.out.println("\nCrashed Processes Are: \n");
          for (int i = 0; i < n; i++) {
            if (prc[i] == 0) System.out.println(i + 1);
            crash++;
          }
          System.out.println("Enter The Process You Want To Recover");
          int rp = Integer.parseInt(br.readLine());
          if ((rp < 1) || (rp > n)) System.out.println(
            "\nInvalid Process. Enter A Valid ID"
          ); else if ((prc[rp - 1] == 0) && (rp > cood)) {
            prc[rp - 1] = 1;
            System.out.println("\nProcess " + rp + " Has Recovered");
            cood = rp;
            System.out.println("\nProcess " + rp + " Is The New Coordinator");
          } else if (crash == n) {
            prc[rp - 1] = 1;
            cood = rp;
            System.out.println("\nProcess " + rp + " Is The New Coordinator");
            crash--;
          } else if ((prc[rp - 1] == 0) && (rp < cood)) {
            prc[rp - 1] = 1;
            System.out.println("\nProcess " + rp + " Has Recovered");
          } else System.out.println(
            "\nProcess " + rp + " Is Not A Crashed Process"
          );
          break;
```

```
        case 3:
          System.out.println("\nCurrent Coordinator Is " + cood);
          break;
        case 4:
          System.out.println(0);
          break;
        default:
          System.out.println("\nInvalid Entry!");
          break;
      }
    } while (ch != 4);
  }

  public static void main(String args[]) throws IOException {
    BullyAlgo ob = new BullyAlgo();
    ob.Bully();
  }
}
```

Output:

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 5 -- Bully Election Algorithm>java BullyAlgo
Enter The Number Of Processes:
5

        1. Crash A Process
        2. Recover A Process
        3. Display New Cordinator
        4. Exit
1

Enter A Process To Crash
1

Process 1 Has Been Crashed
```

```
        1. Crash A Process
        2. Recover A Process
        3. Display New Cordinator
        4. Exit
3

Current Coordinator Is 5

        1. Crash A Process
        2. Recover A Process
        3. Display New Cordinator
        4. Exit
1

Enter A Process To Crash
5

The Coordinator Has Crashed!

Enter The Intiator
4
Process 4 Called For Election
Process 5 Called For Election

Process 4 Is In
Process 5 Is Dead

*** New Coordinator Is 4 ***

        1. Crash A Process
        2. Recover A Process
        3. Display New Cordinator
        4. Exit
2

Crashed Processes Are:
```

```
1
5
Enter The Process You Want To Recover
5

Process 5 Has Recovered

Process 5 Is The New Coordinator

     1. Crash A Process
     2. Recover A Process
     3. Display New Cordinator
     4. Exit
4
```

Conclusion:
The bully election algorithm is a leader election algorithm for distributed systems. It provides a guaranteed way of determining a leader from a set of nodes that manages key responsibilities in communication. It works by the principle that nodes with higher numbers bully the lower nodes to give up their coordinator role. Thus, the node with the highest process always wins the election.

EXPERIMENT - 9

Aim: Implementation of Ring Election Algorithm

Theory:
Distributed Algorithm is an algorithm that runs on a distributed system. A distributed system is a collection of independent computers that do not share their memory. Each processor has its memory, and they communicate via communication networks. Communication in networks is implemented in a process on one machine communicating with a process on another machine. Many algorithms used in the distributed system require a coordinator that performs functions needed by other processes in the system.

Election algorithms are designed to choose a coordinator.
Election Algorithms: Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes for some reason, a new coordinator is elected on another processor. The election algorithm basically determines where a new copy of the coordinator should be restarted. The election algorithm assumes that every active process in the system has a unique priority number. The process with the highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects the active process that has the highest priority number. Then this number is sent to every active process in the distributed system. We have two election algorithms for two different configurations of a distributed system.

The 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 an active list, a list that has a priority number of all active processes in the system.

Algorithm –

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

If process P2 receives a message elect from processes on left, it responds in 3 ways:

1. If the message received does not contain 1 in the active list then P1 adds 2 to its active list and forwards the message.
2. If this is the first election message it has received or sent, P1 creates a new active list with numbers 1 and 2. It then sends election message 1 followed by 2.
3. If Process P1 receives its own election message 1 then the active list for P1 now contains numbers of all the active processes in the system. Now Process P1 detects the highest priority number from the list and elects it as the new coordinator.

Code:

```
// MutualServer.java

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

public class MutualServer implements Runnable {

  Socket socket = null;
  static ServerSocket ss;

  MutualServer(Socket newSocket) {
    this.socket = newSocket;
  }

  public static void main(String args[]) throws IOException {
    ss = new ServerSocket(7000);
    System.out.println("Server Started");
    while (true) {
      Socket s = ss.accept();
      MutualServer es = new MutualServer(s);
      Thread t = new Thread(es);
      t.start();
    }
  }

  public void run() {
    try {
      BufferedReader in = new BufferedReader(
          new InputStreamReader(socket.getInputStream()));
      while (true) {
        System.out.println(in.readLine());
      }
    } catch (Exception e) {
    }
  }
}
```

```
// ClientOne.java

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

public class ClientOne {

  public static void main(String args[]) throws IOException {
    Socket s = new Socket("localhost", 7000);
    PrintStream out = new PrintStream(s.getOutputStream());
    ServerSocket ss = new ServerSocket(7001);
    Socket s1 = ss.accept();
    BufferedReader in1 = new BufferedReader(
        new InputStreamReader(s1.getInputStream()));
    PrintStream out1 = new PrintStream(s1.getOutputStream());
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String str = "Token";
    while (true) {
      if (str.equalsIgnoreCase("Token")) {
        System.out.println("Do you want to send some data");
        System.out.println("Enter Yes or No");
        str = br.readLine();
        if (str.equalsIgnoreCase("Yes")) {
          System.out.println("Enter the data");
          str = br.readLine();
          out.println(str);
        }
        out1.println("Token");
      }
      System.out.println("Waiting for Token");
      str = in1.readLine();
    }
  }
}
```

```
// ClientTwo.java

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

public class ClientTwo {

  public static void main(String args[]) throws IOException {
    Socket s = new Socket("localhost", 7000);
    PrintStream out = new PrintStream(s.getOutputStream());
    Socket s2 = new Socket("localhost", 7001);
    BufferedReader in2 = new BufferedReader(
        new InputStreamReader(s2.getInputStream()));
    PrintStream out2 = new PrintStream(s2.getOutputStream());
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String str;
    while (true) {
      System.out.println("Waiting for Token");
      str = in2.readLine();
      if (str.equalsIgnoreCase("Token")) {
        System.out.println("Do you want to send some data");
        System.out.println("Enter Yes or No");
        str = br.readLine();
        if (str.equalsIgnoreCase("Yes")) {
          System.out.println("Enter the data");
          str = br.readLine();
          out.println(str);
        }
        out2.println("Token");
      }
    }
  }
}
```

Output:

```
C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 6 -- RIng Algorithm>java MutualServer
Server Started
Data From Client One
Data From Client Two
Data From Client Two again
Data From Client One again
```

```
PS C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 6 -- RIng Algorithm> java ClientOne
Do you want to send some data
Enter Yes or No
Yes
Enter the data
Data From Client One
Waiting for Token
Do you want to send some data
Enter Yes or No
```

```
No
Waiting for Token
Do you want to send some data
Enter Yes or No
Yes
Enter the data
Data From Client One again
Waiting for Token
```

```
PS C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical 6 -- RIng Algorithm> java ClientTwo
Waiting for Token
Do you want to send some data
Enter Yes or No
Yes
Enter the data
Data From Client Two
Waiting for Token
Do you want to send some data
Enter Yes or No
Yes
Enter the data
Data From Client Two again
Waiting for Token
Do you want to send some data
Enter Yes or No
```

Conclusion:
The token ring algorithm ensures mutual exclusion by using a unique token the bearer of which is allowed access to the critical section. It ensures that the token circulates in a fair fashion giving each node equal opportunity and avoiding starvation. It however suffers from the drawback of complex procedures to determine if a token is indeed lost and to regenerate it while avoiding double tokens. The Ricart Agrawala algorithm on the other hand is permission-based and follows a

FIFO access structure regulated by peer nodes. It allows nodes to determine order priority based on logical timestamps thus penalizing heavy accessing nodes.

EXPERIMENT - 10

Aim: Demonstration of Deadlock Management

Theory:

A deadlock is a condition in a system where a set of processes (or threads) have requests for resources that can never be satisfied. Essentially, a process cannot proceed because it needs to obtain a resource held by another process; but it itself is holding a resource that the other process needs.

There are four conditions to be met for a deadlock to occur in a system:

1. Mutual exclusion: A resource can be held by at most one process.
2. Hold and wait: Processes that already hold resources can wait for another resource.
3. Non-preemption: A resource, once granted, cannot be taken away.
4. Circular wait: Two or more processes are waiting for resources held by one of the other processes.

The banker's algorithm is a resource allocation and deadlock avoidance algorithm used in a distributed system. The implementation of the banker's algorithm in Java is as follows:

Code:

```
// Bankers.java

import java.util.Scanner;

public class Bankers {
   private int need[][], allocate[][], max[][], avail[][], np, nr;

   private void input() {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter no. of processes and resources : ");
```

```
    np = sc.nextInt(); // no. of process
    nr = sc.nextInt(); // no. of resources
    need = new int[np][nr]; // initializing arrays
    max = new int[np][nr];
    allocate = new int[np][nr];
    avail = new int[1][nr];

    System.out.println("Enter allocation matrix -->");
    for (int i = 0; i < np; i++)
      for (int j = 0; j < nr; j++)
        allocate[i][j] = sc.nextInt(); // allocation matrix

    System.out.println("Enter max matrix -->");
    for (int i = 0; i < np; i++)
      for (int j = 0; j < nr; j++)
        max[i][j] = sc.nextInt(); // max matrix

    System.out.println("Enter available matrix -->");
    for (int j = 0; j < nr; j++)
      avail[0][j] = sc.nextInt(); // available matrix

    sc.close();
  }

  private int[][] calc_need() {
    for (int i = 0; i < np; i++)
      for (int j = 0; j < nr; j++) // calculating need matrix
        need[i][j] = max[i][j] - allocate[i][j];

    return need;
  }

  private boolean check(int i) {
    // checking if all resources for ith process can be allocated
    for (int j = 0; j < nr; j++)
      if (avail[0][j] < need[i][j])
        return false;
        return true;
    }

    public void isSafe() {
        input();
        calc_need();
        boolean done[] = new boolean[np];
        int j = 0;
        while (j < np) { // until all process allocated
            boolean allocated = false;
            for (int i = 0; i < np; i++)
                if (!done[i] && check(i)) { // trying to allocate
                    for (int k = 0; k < nr; k++)
                        avail[0][k] = avail[0][k] - need[i][k] + max[i][k];
                    System.out.println("Allocated process : " + i);
                    allocated = done[i] = true;
                    j++;
                }
            if (!allocated)
                break; // if no allocation
        }
        if (j == np) // if all processes are allocated
            System.out.println("\nSafely allocated");
        else
            System.out.println("All processes cannot be allocated safely");
    }

    public static void main(String[] args) {
        new Bankers().isSafe();
    }

}
```

Output:

```
PS C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM 7\Distributed Computing\practical X3 -- Bankers Algorithm for Deadlock> java Bankers
```

```
Enter no. of processes and resources : 3 4
Enter allocation matrix -->
1 3 4 2
1 2 3 0
1 2 3 3
Enter max matrix -->
3 2 2 1
2 1 3 4
1 3 4 0
Enter available matrix -->
3 4 1 2
Allocated process : 0
Allocated process : 1
Allocated process : 2

Safely allocated
```

Conclusion:
The banker's algorithm is a reliable deadlock avoidance that can schedule processes based on their resource needs to allow execution of all of the processes and avoid deadlock. This can be extended to schedule orders in the supply chain by managing supply and demand to schedule orders similar to resources in an operating system.

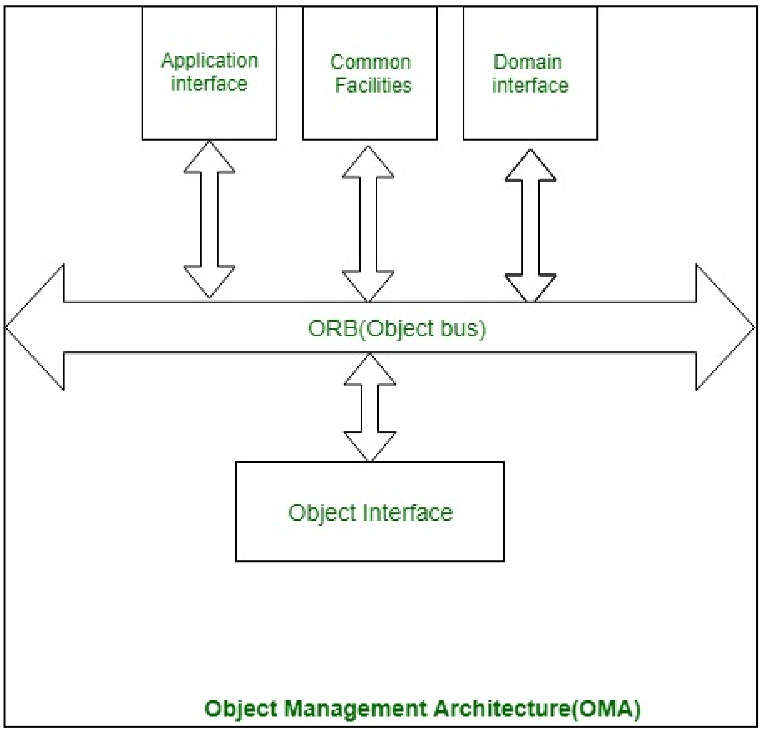
EXPERIMENT - 11

Aim: Demonstration of CORBA

Theory:
Common Object Request Broker Architecture (CORBA) could be a specification of a regular design for middleware. It is a client-server software development model.

Using a CORBA implementation, a shopper will transparently invoke a way on a server object, which may air a similar machine or across a network. The middleware takes the decision associated to blame for finding an object which will implement the request, passing it the parameters, invoking its methodology, and returning the invocation results. The shopper doesn’t need to remember wherever the item is found, its programming language, software package, or the other aspects that don’t seem to be a part of the associated object’s interface.

CORBA Reference Model:
The CORBA reference model, known as Object Management design (OMA) is shown below. The OMA is a specification (actually, a group of connected specifications) that defines various services for building distributed client-server applications. several services one may expect to search out in a very middleware product like CORBA (e.g., naming, dealings, and asynchronous event management services) are literally fixed as services within the OMA



Different parts communicate victimization ORB. ORB is additionally referred to as the item bus. An associate example of the application interface is a distributed document facility. In a very domain interface, it will have domain-dependent services, for instance, producing domain.

Object interface has some domain freelance services:

1. **Naming Service:**
   Naming service is additionally known as a white page service. victimization naming service server name will be searched, and it's location, or address will be pointed out.

2. **Trading Service**:
   Commercialism is also known as a yellow page service. victimization commercialism service, a selected service will be searched. This often corresponds to looking out for a service like an automobile store in a very yellow page directory.

The CORBA Application is composed of three programs:

a. **idl program**– which contains the declaration of methods to be called by the client and defined by the server program. The return type of method or parameters should not be integer as CORBA does not support integer data type; instead short or double can be used.

b. **Server Program** – which contains definition of the methods which are declared in idl file and called by the client program.

c. **Client program** – which contains Method calling defined at the server.

Code:

```
// Hello.idl

module HelloApp{
interface Hello {
   // sayHello() is declared which can be replaced with own method declaration
   string sayHello();oneway

   void shutdown();
};};
```

```
// HelloServer.java

import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;

class HelloImpl extends HelloPOA {
    private ORB orb;

    public void setORB(ORB orb_val) {Ff
        orb = orb_val;
    }

    // implement sayHello() method this definition can be replaced with own method
    public String sayHello() {
        return "\nHello world !!\n";
    }

    // implement shutdown() method
    public void shutdown() {
        orb.shutdown(true);
    }
}

public class HelloServer {
    public static void main(String args[]) {
        try {
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);
            // get reference to rootpoa & activate the POAManager
                                                                    POA       rootpoa       =
POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootpoa.the_POAManager().activate();
            // create servant and register it with the ORB
            HelloImpl helloImpl = new HelloImpl();
            helloImpl.setORB(orb);
            // get object reference from the servant
            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
            Hello href = HelloHelper.narrow(ref);
            // get the root naming context
                                                    org.omg.CORBA.Object     objRef     =
orb.resolve_initial_references("NameService");
```

```
            // Use NamingContextExt which is part of the Interoperable
            // Naming Service (INS) specification.
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
            // bind the Object Reference in Naming
            String name = "Hello";
            NameComponent path[] = ncRef.to_name(name);
            ncRef.rebind(path, href);
            System.out.println("HelloServer ready and waiting ...");
            // wait for invocations from clients
            orb.run();
        } catch (Exception e) {
            System.err.println("ERROR: " + e);
            e.printStackTrace(System.out);
        }
        System.out.println("HelloServer Exiting ...");
    }
}
```

```
// HelloClient.java

import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HelloClient {
    static Hello helloImpl;

    public static void main(String args[]) {
        try {
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);
            // get the root naming context
            org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
            // Use NamingContextExt instead of NamingContext. This is
            // part of the Interoperable naming Service.
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
            // resolve the Object Reference in Naming
            String name = "Hello";
            helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
            System.out.println("Obtained a handle on server object: " + helloImpl);
            System.out.println(helloImpl.sayHello());
            helloImpl.shutdown();
        } catch (Exception e) {
            System.out.println("ERROR : " + e);
            e.printStackTrace(System.out);
        }
    }
}
```

```
start orbd -ORBInitialPort 1050 -ORBInitialHost localhost
```

```
java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost
```

Output:

```
PS C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical X4 -- CORBA> idlj -fall Hello.idl

PS C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM
7\Distributed Computing\Practical X4 -- CORBA> cd HelloApp
```

```
PS C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM 7\Distributed Computing\Practical X4 -- CORBA> javac *.java
```

```
PS C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM 7\Distributed Computing\Practical X4 -- CORBA> java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost
HelloServer ready and waiting ...
```

```
PS C:\Users\JARVIS\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSCE\SEM 7\Distributed Computing\Practical X4 -- CORBA>java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost
Hello World !!!
```

Conclusion:
CORBA is a middleware that allows communication and interfacing between heterogenous systems in a distributed manner. It works by providing a native interface to each of the nodes and performing the complex communication and transfer of data within. It uses an Object Resource Broker that acts as the bridge between the CORBA client and the CORBA server with various applicable call semantics.

EXPERIMENT - 12

Aim: Implementation of Hadoop Distributed File System

Theory:
HDFS(Hadoop Distributed File System) is utilized for storage permission is a Hadoop cluster. It is mainly designed for working on commodity Hardware devices(devices that are inexpensive) and working on a distributed file system design. HDFS is designed in such a way that it believes more in storing the data in a large chunk of blocks rather than storing small data blocks. HDFS in Hadoop provides Fault-tolerance and High availability to the storage layer and the other devices present in that Hadoop cluster.

HDFS is capable of handling larger size data with high volume velocity and variety doing Hadoop work more efficiently and reliably with easy access to all its components. HDFS stores the data in the form of the block where the size of each data block is 128MB in size which is configurable means you can change it according to your requirement in a hdfs-site.xml file in your Hadoop directory.

Some Important Features of HDFS(Hadoop Distributed File System)

- It's easy to access the files stored in HDFS.
- HDFS also provides high availability and fault tolerance.
- Provides scalability to scale up or scaledown nodes as per our requirement.
- Data is stored in distributed manner i.e., various Datanodes are responsible for storing the data.
- HDFS provides Replication because of which there is no fear of Data Loss.
- HDFS Provides High Reliability as it can store data in a large range of Petabytes.
- HDFS has in-built servers in the Name node and Data Node that helps them to retrieve the cluster information easily.
- Provides high throughput.
- HDFS Storage Daemon's

As we all know, Hadoop works on the MapReduce algorithm, which is a master-slave architecture, HDFS has NameNode and DataNode that works in a similar pattern.

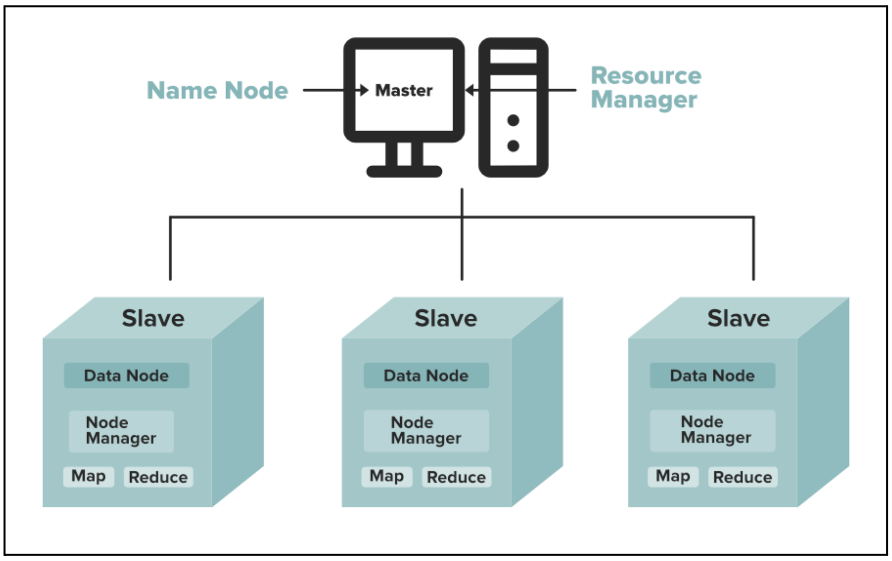
1. NameNode(Master)
2. DataNode(Slave)

1. **NameNode**: NameNode works as a Master in a Hadoop cluster that Guides the Datanode(Slaves). Namenode is mainly used for storing the Metadata i.e. nothing but the data about the data. Meta Data can be the transaction logs that keep track of the user's activity in a Hadoop cluster.

   Meta Data can also be the name of the file, size, and the information about the location(Block number, Block ids) of the Datanode that Namenode stores to find the closest DataNode for Faster Communication. Namenode instructs the DataNodes with operations like delete, create, Replicate, etc.

   As our NameNode is working as a Master, it should have a high RAM or Processing power in order to Maintain or Guide all the slaves in a Hadoop cluster. Namenode receives heartbeat signals and block reports from all the slaves, i.e., DataNodes.

2. **DataNode**: DataNodes work as a Slave DataNodes are mainly utilized for storing the data in a Hadoop cluster; the number of DataNodes can be from 1 to 500 or even more than that, the more number of DataNode your Hadoop cluster has More Data can be stored. so it is advised that the DataNode should have a High storing capacity to store a large number of file blocks. Datanode performs operations like creation, deletion, etc., according to the instruction provided by the NameNode.



Objectives and Assumptions Of HDFS

1. **System Failure**: As a Hadoop cluster is consists of Lots of nodes with are commodity hardware, so node failure is possible, so the fundamental goal of HDFS figure out this failure problem and recover it.

2. **Maintaining Large Dataset**: As HDFS Handles files of size ranging from GB to PB, HDFS has to be cool enough to deal with these very large data sets on a single cluster.

3. **Moving Data is Costlier then Moving the Computation**: If the computational operation is performed near the location where the data is present, then it is quite faster, and the overall throughput of the system can be increased along with minimizing the network congestion, which is a good assumption.

4. **Portable Across Various Platforms**: HDFS Posses portability which allows it to switch across diverse Hardware and software platforms.

5. **Simple Coherency Model**: A Hadoop Distributed File System needs a model to write once read much access for Files. A file written and then closed should not be changed, only data can be appended. This assumption helps us to minimize the data coherency issue. MapReduce fits perfectly with such kind of file model.

HDFS Commands:

1. **ls**: This command is used to list all the files. Use lsr for the recursive approach. It is useful when we want a hierarchy of a folder.
   Syntax:
   ```
   bin/hdfs dfs -ls  <path>
   ```
   Example:
   ```
   bin/hdfs dfs -ls /
   ```
   It will print all the directories present in HDFS. bin directory contains executables so, bin/hdfs means we want the executables of hdfs particularly dfs(Distributed File System) commands.

2. **mkdir**: To create a directory. In Hadoop dfs there is no home directory by default. So let's first create it.
   Syntax:
   ```
   bin/hdfs dfs -mkdir <folder name>
   ```
   creating home directory:
   ```
   hdfs/bin -mkdir /user
   hdfs/bin -mkdir /user/username -> write the username of your computer
   Example:
   ```
   ```
   bin/hdfs dfs -mkdir  /geeks  =>  '/' means absolute path
   bin/hdfs dfs -mkdir  geeks2  =>   Relative path -> the folder will be
                                  created relative to the home directory.
   ```

3. **touchz**: It creates an empty file.

   Syntax:

```
bin/hdfs dfs  -touchz  <file_path>
```

Example:

```
bin/hdfs dfs -touchz  /geeks/myfile.txt
```

4. **copyFromLocal** (or) put: To copy files/folders from local file system to hdfs store. This is the most important command. Local filesystem means the files present on the OS.

   Syntax:

   ```
   bin/hdfs dfs -copyFromLocal <local file path>  <dest(present on hdfs)>
   ```

   Example: Let's suppose we have a file AI.txt on Desktop which we want to copy to folder geeks present on hdfs.

   ```
   bin/hdfs dfs -copyFromLocal ../Desktop/AI.txt /geeks
   ```

   (OR)

   ```
   bin/hdfs dfs -put ../Desktop/AI.txt /geeks
   ```

5. **cat**: To print file contents.

   Syntax:

   ```
   bin/hdfs dfs -cat <path>
   ```

   Example:

   ```
   // print the content of AI.txt present
   // inside geeks folder.
   bin/hdfs dfs -cat /geeks/AI.txt ->
   ```

6. **copyToLocal** (or) get: To copy files/folders from hdfs store to local file system.

   Syntax:

   ```
   bin/hdfs dfs -copyToLocal  <<srcfile(on hdfs)> <local file dest>
   ```

   Example:

   ```
   bin/hdfs dfs -copyToLocal  /geeks   ../Desktop/hero
   ```

   (OR)

   ```
   bin/hdfs dfs -get /geeks/myfile.txt  ../Desktop/hero
   ```

   myfile.txt from geeks folder will be copied to folder hero present on Desktop.

7. **moveFromLocal**: This command will move file from local to hdfs.

   Syntax:

   ```
   bin/hdfs dfs -moveFromLocal <local src>   <dest(on hdfs)>
   ```

   Example:

   ```
   bin/hdfs dfs -moveFromLocal  ../Desktop/cutAndPaste.txt   /geeks
   ```

8. **cp**: This command is used to copy files within hdfs. Lets copy folder geeks to geeks_copied.

   Syntax:

   ```
   bin/hdfs dfs -cp  <src(on hdfs)>  <dest(on hdfs)>
   ```

   Example:

```
bin/hdfs -cp /geeks  /geeks_copied
```

9. mv: This command is used to move files within hdfs. Lets cut-paste a file myfile.txt from geeks folder to geeks_copied.

   Syntax:
   ```
   bin/hdfs dfs -mv  <src(on hdfs)> <src(on hdfs)>
   ```

   Example:
   ```
   bin/hdfs  -mv  /geeks/myfile.txt  /geeks_copied
   ```

10. **rmr**: This command deletes a file from HDFS recursively. It is a very useful command when you want to delete a non-empty directory.

    Syntax:
    ```
    bin/hdfs dfs -rmr <filename/directoryName>
    ```

    Example:
    ```
    bin/hdfs dfs -rmr  /geeks_copied -> It will delete all the content inside the
                                  directory then the directory itself.
    ```

11. **du**: It will give the size of each file in directory.

    Syntax:
    ```
    bin/hdfs dfs -du  <dirName>
    ```

    Example:
    ```
    bin/hdfs dfs -du /geeks
    ```

12. **Dus**: This command will give the total size of directory/file.

    Syntax:
    ```
    bin/hdfs dfs -dus  <dirName>
    ```

    Example:
    ```
    bin/hdfs dfs -dus /geeks
    ```

13. **stat**: It will give the last modified time of directory or path. In short it will give stats of the directory or file.

    Syntax:
    ```
    bin/hdfs  dfs -stat    <hdfs file>
    ```
    Example:
    ```
    bin/hdfs dfs -stat /geeks
    ```

14. **setrep**: This command is used to change the replication factor of a file/directory in HDFS. By default it is 3 for anything which is stored in HDFS (as set in hdfs core-site.xml).

    Example 1: To change the replication factor to 6 for geeks.txt stored in HDFS.
    ```
    bin/hdfs dfs -setrep -R -w 6 geeks.txt
    ```

    Example 2: To change the replication factor to 4 for a directory geeksInput stored in HDFS.
    ```
    bin/hdfs dfs -setrep -R  4 /geeks
    ```
    Note: The -w means wait till the replication is completed. And -R means recursively, we use it for directories as they may also contain many files and folders inside them.

IMPLEMENTATION:

```
[training@localhost ~]$ pwd
```

```
/home/training
```

```
[training@localhost ~]$ ls
```

```
201212daily.txt       KMeans$Map.class         sample5.txt
aaa.txt               KMeans$Reduce.class      sample6.txt
abcde                 km.jar                   sample-bloomfilter
a.txt                 Kpack1 .java             sample-filter
backgrounds           Kpack1.java              sorrt.jar
bb.txt                Kpack1.java~             sortData
BloomFilter.java      kriti                    sortData~
bloom.jar             lib                      SortData
book                  ma                       SortData.class
bookinfo              ma.jar                   SortData.java
bookinfo~             mat                      sorting
Books                 mat1                     sorting1.jar
b.txt                 mat1.jar                 sort.jar
build                 mat2.jar                 SortMapper
Centroid              mat3.jar                 SortMapper.class
Centroid~             mat4.jar                 SortMapper.java
centroids.txt         mat5.jar                 SortReducer
centroid.txt          mat6.jar                 SortReducer.class
Clustering1.java      mat.jar                  SortReducer.java
Clustering1.java~     mat.jar-C                stored
cust_file_class_2009  matrcies1.jar            stored~
Data                  matrices                 student1.java
data1_2009            matrices1.jar            student.java
data1_2009~           matrix                   t
data2_2009            Matrix.class             table1.txt
data2_2009~           matrix.jar               table1.txt~
data3_2009~           matrix.java              table2.txt
data4_2009~           #Matrix.java#            table2.txt~
data.txt              Matrix.java              temp
data.txt~             Matrix.java~             Templates
```

```
deptinfo              Matrix$Map.class         TempStatsStore
derby.log             matrixmuln               test
Desktop               Matrixmulti              test1.txt
dir6                  Matrixmulti.java         test1.txt~
dirbook               Matrixmulti.java~        test.jar
djs                   Matrix$Reduce.class      textfile2.txt
djs.jar               Matrix.txt               textfile3
djss                  matrixx.jar              textfile4
eclipse               matt                     textfile.txt
emp                   mattt                    training_materials
emp~                  MRDemo.jar               tushar
emp2                  new file                 tushar1
emp2~                 new file~                tushar1.jar
emp2.java             new file 1               tushar.jar
emp3.java             new file 2               untitled folder
emp.java              new file 2~              w
emp_name              order_custid             WeatherData
example~              pig_1443861414011.log    w.jar
file1.txt             pig_1443947154908.log    word
file1.txt~            pig_1443949486150.log    Word1.java
filetext4             pig_1444470259655.log    word2.jar
FilterJob.java        pig_1444547817901.log    WordCount12.java
FilterMapper.java     pig_1448259294240.log    wordcount1.jar
gender                pig_1448278570362.log    WordCount.class
hadoop                pig_1448684754725.log    wordcount_classes
HadoopPoem1.txt       pig_1448685352126.log    wordcount.java
HadoopPoem2.txt       poem912                  WordCount.java
inner2.txt            posts.txt                WordCount.java~
inner.txt             product_file_class_2009  WordCount.java~~
input                 proto_file1.txt          WordCount$MapClass.class
java                  proto_file2.txt          WordCount$Reduce.class
kmeans                proto_file3.txt          word.jar
kmeans1.jar           proto_file4.txt          wordtext
KMeans.class          qqq                      workspace
kmeans.jar            QueryResult.java         w.txt
"KMeans.java"         sample                   w.txt~
"KMeans.java"~        sample~                  www.java
KMeans.java           sample1                  wwww.jar
KMeans.java~          sample1~
```

```
[training@localhost ~]$ hadoop dfs -ls
```

```
Found 109 items
-rw-r--r--    1 training supergroup       1390 2015-10-02 20:20
/user/training/Books
drwxr-xr-x    - training supergroup          0 2014-08-17 03:50
/user/training/WeatherData
-rw-r--r--    1 training supergroup       3103 2016-02-16 21:54
/user/training/WordCount.java
drwxr-xr-x    - training supergroup          0 2015-10-17 02:34
/user/training/_sqoop
-rw-r--r--    1 training supergroup          0 2018-02-01 23:46
/user/training/aaa.txt
drwxr-xr-x    - training supergroup          0 2018-02-02 00:45
/user/training/abcdef
-rw-r--r--    1 training supergroup         12 2020-09-25 00:58
/user/training/asd
-rw-r--r--    1 training supergroup         18 2018-02-01 23:52
/user/training/bb.txt
drwxr-xr-x    - training supergroup          0 2016-03-09 22:55
/user/training/bbb11451728037137
drwxr-xr-x    - training supergroup          0 2016-03-10 01:05
/user/training/bbbb17422138347695
drwxr-xr-x    - training supergroup          0 2016-03-10 01:06
/user/training/bbbb17569614160750
drwxr-xr-x    - training supergroup          0 2016-03-10 01:15
/user/training/bbbb18073723471225
-rw-r--r--    1 training supergroup       2944 2015-09-26 02:37
/user/training/bookinfo
drwxr-xr-x    - training supergroup          0 2015-09-20 01:38
/user/training/class2009_dir1
-rw-r--r--    1 training supergroup         81 2015-09-18 21:05
/user/training/deptinfo
drwxr-xr-x    - training supergroup          0 2014-08-17 01:59
/user/training/dir1
drwxr-xr-x    - training supergroup          0 2015-09-12 02:42
/user/training/dir10
drwxr-xr-x    - training supergroup          0 2015-09-19 02:00
/user/training/dir100
```

```
drwxr-xr-x     - training  supergroup            0  2015-09-19  02:42
/user/training/dir1000
drwxr-xr-x     - training  supergroup            0  2014-08-22  02:30
/user/training/dir11
drwxr-xr-x     - training  supergroup            0  2014-08-22  02:35
/user/training/dir12
drwxr-xr-x     - training  supergroup            0  2014-08-17  01:59
/user/training/dir2
drwxr-xr-x     - training  supergroup            0  2014-08-17  01:59
/user/training/dir3
drwxr-xr-x     - training  supergroup            0  2014-08-17  01:59
/user/training/dir4
drwxr-xr-x     - training  supergroup            0  2014-08-17  01:59
/user/training/dir5
drwxr-xr-x     - training  supergroup            0  2014-08-17  03:03
/user/training/dir6
drwxr-xr-x     - training  supergroup            0  2014-08-17  03:04
/user/training/dir7
drwxr-xr-x     - training  supergroup            0  2014-08-17  04:22
/user/training/dir9
-rw-r--r--     1  training  supergroup        1390  2015-10-02  20:22
/user/training/dirbook
drwxr-xr-x     - training  supergroup            0  2021-10-05  01:06
/user/training/djsin
drwxr-xr-x     - training  supergroup            0  2021-10-05  01:06
/user/training/djsout
drwxr-xr-x     - training  supergroup            0  2021-10-05  01:16
/user/training/djsout1
drwxr-xr-x     - training  supergroup            0  2021-10-05  01:12
/user/training/djss
drwxr-xr-x     - training  supergroup            0  2015-10-17  02:14
/user/training/emp
drwxr-xr-x     - training  supergroup            0  2015-10-17  02:34
/user/training/emp2
drwxr-xr-x     - training  supergroup            0  2015-10-10  23:18
/user/training/emp_dir
drwxr-xr-x     - training  supergroup            0  2015-10-17  02:26
/user/training/employee
drwxr-xr-x     - training  supergroup            0  2015-10-17  23:39
```

```
/user/training/employee2
drwxr-xr-x   - training supergroup          0 2015-10-17 23:42
/user/training/employee3
drwxr-xr-x   - training supergroup          0 2015-11-27 20:53
/user/training/hadoop
drwxr-xr-x   - training supergroup          0 2021-07-30 02:17
/user/training/hadoopinword
drwxr-xr-x   - training supergroup          0 2021-07-30 02:12
/user/training/hadoopword
drwxr-xr-x   - training supergroup          0 2021-07-30 02:21
/user/training/hadoopword1
drwxr-xr-x   - training supergroup          0 2016-02-16 02:30
/user/training/imput1234
-rw-r--r--   1 root     supergroup         13 2017-03-10 23:22
/user/training/inword
-rw-r--r--   1 training supergroup         69 2015-10-10 02:35
/user/training/join
-rw-r--r--   1 training supergroup        105 2015-10-10 02:38
/user/training/join2
drwxr-xr-x   - training supergroup          0 2016-03-09 22:47
/user/training/kmeansinput
drwxr-xr-x   - training supergroup          0 2016-03-09 22:42
/user/training/kmeansoutput
drwxr-xr-x   - training supergroup          0 2016-03-09 22:48
/user/training/kmeansoutput10970575692823
drwxr-xr-x   - training supergroup          0 2016-03-08 22:12
/user/training/kminput
drwxr-xr-x   - training supergroup          0 2016-03-08 22:13
/user/training/kmoutput
-rw-r--r--   1 root     supergroup        235 2017-03-10 23:44
/user/training/ma
drwxr-xr-x   - training supergroup          0 2021-10-12 00:14
/user/training/mat1in
drwxr-xr-x   - training supergroup          0 2021-10-12 00:12
/user/training/mat1out
drwxr-xr-x   - training supergroup          0 2021-10-12 00:41
/user/training/mat1out1
drwxr-xr-x   - training supergroup          0 2021-10-13 02:35
/user/training/matricesin
```

```
drwxr-xr-x   - training supergroup          0 2021-10-13 02:33
/user/training/matricesout
drwxr-xr-x   - training supergroup          0 2021-10-13 02:37
/user/training/matricesout1
drwxr-xr-x   - training supergroup          0 2016-02-23 00:16
/user/training/matrixinput
drwxr-xr-x   - training supergroup          0 2016-02-23 00:17
/user/training/matrixoutput
drwxr-xr-x   - training supergroup          0 2021-10-12 01:09
/user/training/mattin
drwxr-xr-x   - training supergroup          0 2021-10-12 01:05
/user/training/mattout
drwxr-xr-x   - training supergroup          0 2021-10-12 01:11
/user/training/mattout1
drwxr-xr-x   - training supergroup          0 2021-10-12 01:21
/user/training/matttin
drwxr-xr-x   - training supergroup          0 2021-10-12 01:20
/user/training/matttout
drwxr-xr-x   - training supergroup          0 2021-10-12 01:22
/user/training/matttout1
-rw-r--r--   1 training supergroup        235 2017-03-17 01:44
/user/training/mm
drwxr-xr-x   - training supergroup          0 2020-09-25 02:00
/user/training/out1
drwxr-xr-x   - training supergroup          0 2021-10-13 03:18
/user/training/out25
drwxr-xr-x   - training supergroup          0 2016-02-16 02:32
/user/training/output1234
drwxr-xr-x   - training supergroup          0 2017-03-10 23:27
/user/training/oword2
-rw-r--r--   1 training supergroup         16 2015-10-04 01:23
/user/training/pig
-rw-r--r--   1 training supergroup         32 2015-10-04 02:04
/user/training/pig1
-rw-r--r--   1 training supergroup         90 2015-11-23 03:12
/user/training/poem
-rw-r--r--   1 training supergroup         90 2015-09-12 02:41
/user/training/poem912
drwxr-xr-x   - training supergroup          0 2021-10-13 03:02
```

```
/user/training/sortingin
drwxr-xr-x     - training supergroup          0 2021-10-13 03:00
/user/training/sortingout
drwxr-xr-x     - training supergroup          0 2021-10-13 03:05
/user/training/sortingout1
drwxr-xr-x     - training supergroup          0 2021-10-13 03:08
/user/training/sortingout2
drwxr-xr-x     - training supergroup          0 2021-10-13 03:12
/user/training/sortingout5
drwxr-xr-x     - training supergroup          0 2016-02-22 22:34
/user/training/sortinput
drwxr-xr-x     - training supergroup          0 2021-10-13 03:17
/user/training/sortinputf
drwxr-xr-x     - training supergroup          0 2021-08-28 00:36
/user/training/sortout
drwxr-xr-x     - training supergroup          0 2016-02-22 22:34
/user/training/sortoutput
drwxr-xr-x     - training supergroup          0 2016-02-22 22:35
/user/training/sortoutput2
drwxr-xr-x     - training supergroup          0 2017-03-03 21:50
/user/training/sortoutputtt
drwxr-xr-x     - training supergroup          0 2021-08-28 01:04
/user/training/ss1
drwxr-xr-x     - training supergroup          0 2021-08-28 01:03
/user/training/sss
drwxr-xr-x     - training supergroup          0 2015-10-10 03:53
/user/training/str
drwxr-xr-x     - training supergroup          0 2015-10-17 01:26
/user/training/stud
drwxr-xr-x     - training supergroup          0 2015-10-17 01:42
/user/training/stud1
drwxr-xr-x     - training supergroup          0 2015-10-17 01:43
/user/training/stud2
drwxr-xr-x     - training supergroup          0 2015-10-17 02:01
/user/training/student
-rw-r--r--     1 training supergroup         86 2015-10-10 03:40
/user/training/table2
drwxr-xr-x     - training supergroup          0 2016-02-05 19:51
/user/training/test
```

```
drwxr-xr-x     -  training  supergroup             0  2021-10-05  01:48
/user/training/test1
-rw-r--r--     1  training  supergroup            29  2021-10-05  01:54
/user/training/test2
drwxr-xr-x     -  training  supergroup             0  2021-10-05  01:51
/user/training/testout
-rw-r--r--     1  training  supergroup            24  2016-02-12  00:40
/user/training/textfile.txt
drwxr-xr-x     -  training  supergroup             0  2020-11-07  00:08
/user/training/tushar1input
drwxr-xr-x     -  training  supergroup             0  2020-11-07  00:13
/user/training/tushar1output
drwxr-xr-x     -  training  supergroup             0  2020-11-07  00:24
/user/training/tushar1output9
drwxr-xr-x     -  training  supergroup             0  2020-11-06  22:17
/user/training/tusharinput
drwxr-xr-x     -  training  supergroup             0  2020-11-06  22:12
/user/training/tusharoutput
drwxr-xr-x     -  training  supergroup             0  2020-11-06  22:14
/user/training/tusharoutput1
drwxr-xr-x     -  training  supergroup             0  2020-11-06  22:17
/user/training/tusharoutput2
drwxr-xr-x     -  training  supergroup             0  2015-09-12  02:33
/user/training/user
-rw-r--r--     1  training  supergroup            59  2015-10-10  00:56
/user/training/wordcount
```

```
[training@localhost ~]$ hadoop dfs -mkdir hadoop
[training@localhost ~]$ hadoop dfs -cp /user/training/WordCount.java
/user/training/hadoop
[training@localhost ~]$ hadoop dfs -touchz rem.txt
[training@localhost ~]$ hadoop dfs -mv /user/training/rem.txt
/user/training/hadoop
[training@localhost ~]$ hadoop dfs -mkdir hadoop2
[training@localhost ~]$ hadoop dfs -cp /user/training/hadoop/rem.text
/user/training/hadoop2
```

```
cp: File does not exist: /user/training/hadoop/rem.text
```

```
[training@localhost  ~]$  hadoop  dfs  -cp  /user/training/hadoop/rem.txt
/user/training/hadoop2
[training@localhost ~]$ hadoop dfs -rm
[training@localhost ~]$ hadoop dfs -rm /user/training/hadoop2/rem.txt
```

```
Deleted hdfs://localhost/user/training/hadoop2/rem.txt
```

```
[training@localhost ~]$ hadoop dfs -rm -rf /user/training/hadoop2
```

```
rm: cannot remove -rf: No such file or directory.
rm:  Cannot  remove  directory  "hdfs://localhost/user/training/hadoop2",
use -rmr instead
```

```
[training@localhost ~]$ hadoop dfs -rmr /user/training/hadoop2
```

```
Deleted hdfs://localhost/user/training/hadoop2
```

```
[training@localhost ~]$ hadoop dfs -cat /user/training/bb.txt
```

```
jhhhl
kjhkjh
laks
```

```
[training@localhost ~]$ hadoop dfs -cp /user/training/WordCount.java

/user/training/hadoop
[training@localhost ~]$ hadoop dfs -mv /user/training/WordCount.java
```

```
/user/training/hadoop
mv:  Failed  to  rename  hdfs://localhost/user/training/WordCount.java  to
/user/training/hadoop
```

```
[training@localhost  ~]$  hadoop  dfs  -copyFromLocal  /home/training/aaa.txt
/user/training/hadoop
```

```
[training@localhost       ~]$       hadoop       dfs       -copyToLocal
/user/training/hadoop/WordCount.java  /home/training
```

```
copyToLocal: Target /home/training/WordCount.java already exists
```

```
[training@localhost       ~]$       hadoop       dfs       -moveToLocal
/user/training/hadoop/WordCount.java  /home/training
```

```
Option '-moveToLocal' is not implemented yet.
```

```
[training@localhost       ~]$       hadoop       dfs       -moveFromLocal
/user/training/hadoop/WordCount.java  /home/training
moveFromLocal: File /user/training/hadoop/WordCount.java does not exist.
```

```
[training@localhost ~]$ hadoop dfs -moveFromLocal  /home/training/bb.txt
/user/training/hadoop
```

```
[training@localhost       ~]$       hadoop       dfs       -copyToLocal
/user/training/hadoop/WordCount.java  /home/training
```

```
copyToLocal: Target /home/training/WordCount.java already exists
```

```
[training@localhost ~]$ hadoop dfs -mkdir hadoop2
[training@localhost ~]$ hadoop dfs -copyFromLocal /home/training/aaa.txt
/user/training/hadoop2
```

```
[training@localhost ~]$ hadoop dfs -rm r
```

```
rm: cannot remove r: No such file or directory.
```

```
[training@localhost ~]$ hadoop dfs -rmr hadoop2
```

```
Deleted hdfs://localhost/user/training/hadoop2
```

```
[training@localhost ~]$ hadoop dfs -touchz abc.txt
[training@localhost    ~]$    hadoop    dfs    -put    /home/training/a.txt
/user/training
[training@localhost ~]$ hadoop dfs -rmr abc.txt
```

```
Deleted hdfs://localhost/user/training/abc.txt
```

```
[training@localhost ~]$ hadoop dfs -rm aaa.txt
```

```
Deleted hdfs://localhost/user/training/aaa.txt
```

Conclusion:
Hadoop Distributed File System is a storage layer that allows secure and reliable storage for the Hadoop framework in a distributed manner. It uses a number of nodes to distribute the files and metadata across the network to enable faster response time and security. It also performs many layers of replication generally following the 3-2-1 strategy to ensure data reliability and availability.