**Code 1:**

Implement multi-threaded client/server Process communication using RMI.

**FileName: HelloInterface.java**

import java.rmi.Remote;

import java.rmi.RemoteException;

public interface HelloInterface extends Remote {

    String say(String clientName) throws RemoteException;

}

**FileName : Hello.java**

import java.rmi.server.UnicastRemoteObject;

import java.rmi.RemoteException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Hello extends UnicastRemoteObject implements HelloInterface {

    public Hello() throws RemoteException {

        super(); // Export object on anonymous port

    }

    // Each client will call this with their name

    public String say(String clientName) throws RemoteException {

        String time = new SimpleDateFormat("HH:mm:ss").format(new Date());

        System.out.println("Client " + clientName + " invoked say() at " + time);

        return "Hello, " + clientName + "! [Server time: " + time + "]";

    }

}

**FileName : HelloServer.java**

import java.rmi.Naming;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

public class HelloServer {

    public static void main(String[] args) {

        try {

            // Start RMI Registry

            LocateRegistry.createRegistry(1099); // default port

            Hello obj = new Hello();

            Naming.rebind("HelloService", obj);

            System.out.println("HelloServer is ready.");

        } catch (RemoteException re) {

            System.err.println("RemoteException: " + re);

        } catch (Exception e) {

            System.err.println("Exception: " + e);

        }

    }

}

**FileName : HelloClient.java**

import java.rmi.Naming;

public class HelloClient {

    static class ClientWorker extends Thread {

        private String name;

        public ClientWorker(String name) {

            this.name = name;

        }

        public void run() {

            try {

                HelloInterface stub = (HelloInterface) Naming.lookup("//localhost/HelloService");

                String response = stub.say(name);

                System.out.println("Response for " + name + ": " + response);

            } catch (Exception e) {

                System.err.println("Exception for " + name + ": " + e);

            }

        }

    }

    public static void main(String[] args) {

        // Simulate 5 concurrent clients

        for (int i = 1; i <= 5; i++) {

            new ClientWorker("Client" + i).start();

        }

    }

}

**COMPILATION AND EXECUTION STEPS:**

Command 1: javac \*.java

Command 2: java HelloServer

Command 3: java HelloClient

**Code 2:**

develop any distributed application using CORBA to demonstrate object brokering (calculator or String operations)  
  
Step 1: Write the IDL File

**File: StringApp.idl**

module StringApp {

interface StringOperations {

string toUpper(in string str);

string toLower(in string str);

string reverse(in string str);

};

};

Step 2: Compile the IDL file using idlj

Command: idlj -fall StringApp.idl  
  
Step 3: Implement the Servant (Server-side Implementation)

File: StringImpl.java

import StringApp.StringOperationsPOA;

public class StringImpl extends StringOperationsPOA {

public String toUpper(String str) {

return str.toUpperCase();

}

public String toLower(String str) {

return str.toLowerCase();

}

public String reverse(String str) {

return new StringBuilder(str).reverse().toString();

}

}

**Step 4: Create the Server**

**File: Server.java**

import org.omg.CORBA.\*;

import org.omg.CosNaming.\*;

import org.omg.PortableServer.\*;

import StringApp.StringOperations;

import StringApp.StringOperationsHelper;

public class Server {

public static void main(String args[]) {

try {

ORB orb = ORB.init(args, null);

POA rootpoa = POAHelper.narrow(orb.resolve\_initial\_references("RootPOA"));

rootpoa.the\_POAManager().activate();

StringImpl stringImpl = new StringImpl();

org.omg.CORBA.Object ref = rootpoa.servant\_to\_reference(stringImpl);

StringOperations href = StringOperationsHelper.narrow(ref);

org.omg.CORBA.Object objRef = orb.resolve\_initial\_references("NameService");

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

NameComponent path[] = ncRef.to\_name("StringOperations");

ncRef.rebind(path, href);

System.out.println("Server ready and waiting...");

orb.run();

} catch (Exception e) {

System.err.println("Server Error: " + e);

e.printStackTrace();

}

}

}

**Step 5: Create the Client**

**File: Client.java**

import StringApp.StringOperations;

import StringApp.StringOperationsHelper;

import org.omg.CORBA.\*;

import org.omg.CosNaming.\*;

public class Client {

public static void main(String args[]) {

try {

ORB orb = ORB.init(args, null);

org.omg.CORBA.Object objRef = orb.resolve\_initial\_references("NameService");

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

StringOperations strObj = StringOperationsHelper.narrow(ncRef.resolve\_str("StringOperations"));

System.out.println("Calling toUpper(\"hello\") → " + strObj.toUpper("hello"));

System.out.println("Calling toLower(\"WORLD\") → " + strObj.toLower("WORLD"));

System.out.println("Calling reverse(\"CORBA\") → " + strObj.reverse("CORBA"));

} catch (Exception e) {

System.err.println("ERROR : " + e);

e.printStackTrace(System.out);

}

}

}

**Step 6: Compile all Java files**

Command: javac \*.java StringApp/\*.java

**Step 7: Start the ORB Daemon (Naming Service)**

(Windows Powershall):  
Command: Start-Process orbd -ArgumentList "-ORBInitialPort 1050 -ORBInitialHost localhost"

**Step 8: Start the Server**

Open a **new terminal**, go to project folder, run:

Command: java Server -ORBInitialPort 1050 -ORBInitialHost localhost

**Step 9: Start the Client**

Open another **new terminal**, run:

Command: java Client -ORBInitialPort 1050 -ORBInitialHost localhost

**Code 3:**

develop a distributed system, to find sum of N elements in an array by distributing N/n elements to n numbers of processors MPI or openMP. Demonstrated by displaying the intermediate sum calculated at different processors.  
  
Filename : ParallelSum.java:

import java.util.Random;

public class ParallelSum {

static final int N = 100; // Total number of elements in the array

static final int NUM\_THREADS = 4; // Number of threads (processors)

// Array to store the values

static int[] array = new int[N];

// Array to store partial sums computed by each thread

static int[] partialSums = new int[NUM\_THREADS];

// Task for each thread: calculate the sum of a chunk of the array

static class SumTask extends Thread {

private int start;

private int end;

private int threadIndex;

public SumTask(int start, int end, int threadIndex) {

this.start = start;

this.end = end;

this.threadIndex = threadIndex;

}

@Override

public void run() {

int sum = 0;

for (int i = start; i < end; i++) {

sum += array[i];

}

// Store the partial sum for this thread

partialSums[threadIndex] = sum;

System.out.println("Thread " + threadIndex + " partial sum: " + sum);

}

}

public static void main(String[] args) {

// Initialize the array with random values between 0 and 99

Random rand = new Random();

for (int i = 0; i < N; i++) {

array[i] = rand.nextInt(100); // Random values between 0 and 99

}

// Divide the work between the threads

int chunkSize = N / NUM\_THREADS;

Thread[] threads = new Thread[NUM\_THREADS];

// Start threads for summing chunks of the array

for (int i = 0; i < NUM\_THREADS; i++) {

int start = i \* chunkSize;

int end = (i == NUM\_THREADS - 1) ? N : (i + 1) \* chunkSize; // Last thread handles any leftover elements

threads[i] = new SumTask(start, end, i);

threads[i].start();

}

// Wait for all threads to finish

try {

for (int i = 0; i < NUM\_THREADS; i++) {

threads[i].join();

}

} catch (InterruptedException e) {

e.printStackTrace();

}

// Calculate the total sum by aggregating the partial sums

int totalSum = 0;

for (int sum : partialSums) {

totalSum += sum;

}

// Display the total sum

System.out.println("Total sum of the array: " + totalSum);

}

}

**Step 1: Compile the Program**

Command: javac ParallelSum.java

**Step 2: Run the Program**

Command: java ParallelSum

Code 4:

implement berkeley algorithm for clock synchronization

Filename : BerkeleyCoordinator.java

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class BerkeleyCoordinator {

private static final int PORT = 12345; // Port for communication

private static final int CLIENT\_COUNT = 3; // Number of clients

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(PORT)) {

System.out.println("Coordinator started, waiting for clients...");

// List to store the client sockets

List<Socket> clientSockets = new ArrayList<>();

// Accept connections from clients

for (int i = 0; i < CLIENT\_COUNT; i++) {

Socket clientSocket = serverSocket.accept();

System.out.println("Client " + (i + 1) + " connected.");

clientSockets.add(clientSocket);

}

// Get times from clients and compute average

List<Long> clientTimes = new ArrayList<>();

for (Socket clientSocket : clientSockets) {

DataInputStream in = new DataInputStream(clientSocket.getInputStream());

long clientTime = in.readLong();

clientTimes.add(clientTime);

System.out.println("Received time from client: " + clientTime);

}

// Calculate average time

long sum = 0;

for (long time : clientTimes) {

sum += time;

}

long averageTime = sum / clientTimes.size();

System.out.println("Average time calculated by coordinator: " + averageTime);

// Send offset back to each client

for (int i = 0; i < CLIENT\_COUNT; i++) {

long offset = averageTime - clientTimes.get(i);

Socket clientSocket = clientSockets.get(i);

DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream());

out.writeLong(offset);

System.out.println("Sending offset to client " + (i + 1) + ": " + offset);

}

// Close client connections

for (Socket clientSocket : clientSockets) {

clientSocket.close();

}

System.out.println("Clock synchronization completed.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

Filename : BerkeleyClient.java

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class BerkeleyClient {

private static final String COORDINATOR\_HOST = "localhost"; // Coordinator's host

private static final int COORDINATOR\_PORT = 12345; // Coordinator's port

public static void main(String[] args) {

try (Socket socket = new Socket(COORDINATOR\_HOST, COORDINATOR\_PORT)) {

System.out.println("Client started, sending local time to coordinator...");

// Simulating a clock with random drift

Random random = new Random();

long localTime = System.currentTimeMillis() + random.nextInt(10000); // Adding some random drift

System.out.println("Client's local time: " + localTime);

// Send the local time to the coordinator

DataOutputStream out = new DataOutputStream(socket.getOutputStream());

out.writeLong(localTime);

// Wait for the time offset from the coordinator

DataInputStream in = new DataInputStream(socket.getInputStream());

long offset = in.readLong();

System.out.println("Received offset from coordinator: " + offset);

// Adjust local time by the offset

long synchronizedTime = localTime + offset;

System.out.println("Adjusted time: " + synchronizedTime);

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Steps to Compile and Run**:

Run the following commands to compile both files:

**javac BerkeleyCoordinator.java**

**javac BerkeleyClient.java**

**Run the Coordinator**:

Start the coordinator first by running:

**java BerkeleyCoordinator**

**Run the Clients**:

java BerkeleyClient

java BerkeleyClient

java BerkeleyClient

Code 5:

**FileName : Process.java**

public class Process extends Thread {

    private int id;

    private boolean hasToken;

    private boolean wantToEnterCS;

    private Process next;

    private TokenRing controller;

    public Process(int id, TokenRing controller) {

        this.id = id;

        this.controller = controller;

        this.hasToken = false;

        this.wantToEnterCS = false;

    }

    public void setNext(Process next) {

        this.next = next;

    }

    public void receiveToken() {

        if (controller.shouldTerminate()) return;

        hasToken = true;

        wantToEnterCS = true;  // Auto request CS when receiving token

        if (wantToEnterCS) {

            enterCriticalSection();

            wantToEnterCS = false;

        }

        controller.incrementTokenPasses();

        passToken();

    }

    public void enterCriticalSection() {

        System.out.println("Process " + id + " entering critical section.");

        try {

            Thread.sleep(500);  // Simulate some work

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

        System.out.println("Process " + id + " leaving critical section.");

    }

    public void passToken() {

        hasToken = false;

        if (!controller.shouldTerminate()) {

            System.out.println("Process " + id + " passing token to Process " + next.id);

            next.receiveToken();

        } else {

            System.out.println("Maximum token passes reached. Terminating simulation.");

            System.exit(0);

        }

    }

    @Override

    public void run() {

        // No need to loop, all logic handled by token passing

    }

}

**Filename : TokenRing.java**

public class TokenRing {

    private static final int MAX\_TOKEN\_PASSES = 6;

    private int tokenPasses = 0;

    public synchronized void incrementTokenPasses() {

        tokenPasses++;

    }

    public synchronized boolean shouldTerminate() {

        return tokenPasses >= MAX\_TOKEN\_PASSES;

    }

    public static void main(String[] args) {

        int n = 6;  // Fixed 6 processes

        TokenRing controller = new TokenRing();

        Process[] processes = new Process[n];

        // Create processes

        for (int i = 0; i < n; i++) {

            processes[i] = new Process(i, controller);

        }

        // Link processes in ring

        for (int i = 0; i < n; i++) {

            processes[i].setNext(processes[(i + 1) % n]);

        }

        // Start all processes

        for (int i = 0; i < n; i++) {

            processes[i].start();

        }

        // Give initial token to process 0

        processes[0].receiveToken();

    }

}

Compile and Run Commands:

**javac Process.java TokenRing.java**

**java TokenRing**

**Code 6:**

Implement the Bully and ring algorithm for leader selection:

**File name: BullyElection.java**

import java.util.Scanner;

class Process {

    int id;

    boolean isActive;

    Process(int id) {

        this.id = id;

        this.isActive = true;

    }

}

public class BullyElection {

    static Process[] processes;

    static int coordinator = -1;

    public static void initiateElection(int initiatorId) {

        System.out.println("\nProcess " + initiatorId + " initiates election...");

        boolean higherResponded = false;

        for (int i = initiatorId + 1; i < processes.length; i++) {

            if (processes[i].isActive) {

                System.out.println("Process " + initiatorId + " sends ELECTION to Process " + i);

                System.out.println("Process " + i + " responds OK");

                higherResponded = true;

            }

        }

        if (!higherResponded) {

            coordinator = initiatorId;

            System.out.println("Process " + initiatorId + " becomes coordinator.");

            broadcastCoordinator();

        } else {

            for (int i = processes.length - 1; i > initiatorId; i--) {

                if (processes[i].isActive) {

                    initiateElection(i);

                    break;

                }

            }

        }

    }

    public static void broadcastCoordinator() {

        for (int i = 0; i < processes.length; i++) {

            if (i != coordinator && processes[i].isActive)

                System.out.println("Process " + coordinator + " informs Process " + i + " that it is the new coordinator.");

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of processes: ");

        int n = sc.nextInt();

        processes = new Process[n];

        for (int i = 0; i < n; i++) {

            processes[i] = new Process(i);

        }

        System.out.print("Enter coordinator process ID: ");

        coordinator = sc.nextInt();

        System.out.print("Enter process ID to crash: ");

        int crashedId = sc.nextInt();

        processes[crashedId].isActive = false;

        System.out.print("Enter initiator process ID for election: ");

        int initiatorId = sc.nextInt();

        initiateElection(initiatorId);

        sc.close();

    }

}

**FileName: RingElection.java**

import java.util.\*;

class RingProcess {

    int id;

    boolean isActive;

    RingProcess(int id) {

        this.id = id;

        this.isActive = true;

    }

}

public class RingElection {

    static RingProcess[] processes;

    static int n;

    public static void initiateElection(int initiatorId) {

        System.out.println("\nProcess " + initiatorId + " initiates RING election...");

        List<Integer> message = new ArrayList<>();

        int current = initiatorId;

        do {

            if (processes[current].isActive) {

                message.add(processes[current].id);

                System.out.println("Election message passed through Process " + current);

            }

            current = (current + 1) % n;

        } while (current != initiatorId);

        int newCoordinator = Collections.max(message);

        System.out.println("New coordinator elected: Process " + newCoordinator);

        int notify = (initiatorId + 1) % n;

        do {

            if (processes[notify].isActive) {

                System.out.println("Coordinator message passed to Process " + notify);

            }

            notify = (notify + 1) % n;

        } while (notify != initiatorId);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of processes: ");

        n = sc.nextInt();

        processes = new RingProcess[n];

        for (int i = 0; i < n; i++) {

            processes[i] = new RingProcess(i);

        }

        System.out.print("Enter process ID to crash: ");

        int crashed = sc.nextInt();

        processes[crashed].isActive = false;

        System.out.print("Enter initiator process ID: ");

        int initiator = sc.nextInt();

        initiateElection(initiator);

        sc.close();

    }

}

**Compilation & Execution Steps:**

# Compile both files

**javac BullyElection.java**

**javac RingElection.java**

# Run the Bully algorithm

**java BullyElection**

# Run the Ring algorithm

**java RingElection**

**Code 7:**

Write a web service and write any distributed application to consume the web service

**File Name: SimpleWebService.java**

import com.sun.net.httpserver.HttpServer;

import com.sun.net.httpserver.HttpExchange;

import com.sun.net.httpserver.HttpHandler;

import java.io.IOException;

import java.io.OutputStream;

import java.net.InetSocketAddress;

public class SimpleWebService {

public static void main(String[] args) throws IOException {

HttpServer server = HttpServer.create(new InetSocketAddress(8000), 0);

server.createContext("/greet", new GreetHandler());

server.setExecutor(null); // creates a default executor

System.out.println("Server started at http://localhost:8000/greet");

server.start();

}

static class GreetHandler implements HttpHandler {

@Override

public void handle(HttpExchange exchange) throws IOException {

String query = exchange.getRequestURI().getQuery();

String name = "Guest";

if (query != null && query.contains("name=")) {

name = query.split("name=")[1];

}

String response = "{\"message\": \"Hello, " + name + "!\"}";

exchange.getResponseHeaders().add("Content-Type", "application/json");

exchange.sendResponseHeaders(200, response.length());

OutputStream os = exchange.getResponseBody();

os.write(response.getBytes());

os.close();

}

}

}

**Compile and Run:**

**javac SimpleWebService.java**

**java SimpleWebService**

**FileName : WebServiceClient.java**

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.InputStream;

import java.net.HttpURLConnection;

import java.net.URL;

public class WebServiceClient {

public static void main(String[] args) {

try {

String name = "Tejas";

URL url = new URL("http://localhost:8000/greet?name=" + name);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("GET");

conn.setRequestProperty("Accept", "application/json");

if (conn.getResponseCode() != 200) {

throw new RuntimeException("HTTP GET Request Failed with Error code : "

+ conn.getResponseCode());

}

InputStream is = conn.getInputStream();

BufferedReader br = new BufferedReader(new InputStreamReader(is));

String output;

StringBuilder responseBuilder = new StringBuilder();

while ((output = br.readLine()) != null) {

responseBuilder.append(output);

}

conn.disconnect();

String jsonResponse = responseBuilder.toString();

// Simple manual JSON parsing (avoid dependency)

String message = jsonResponse.split(":")[1].replace("}", "").replace("\"", "").trim();

System.out.println("Server Response: " + message);

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Compile and Run:**

**javac WebServiceClient.java**

**java WebServiceClient**