Thadomal Shahani Engineering College

Bandra (W.), Mumbai - 400 050.

CERTIFICATE

Certify that Mr./~~Mis~~ Aradhya Talan

of Comp. Engg. Department, Semester 8 with

Roll No. 2003175 has completed a course of the necessary

experiments in the subject Distributed Computing under my

supervision in the **Thadomal Shahani Engineering College**

Laboratory in the year 2023 - 2024

Teacher In- Charge

Head of the Department

Date 22/4/24

Principal

THADOMAL SHAHANI TSEC ENGINEERING SOLLEGE

CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1.	WAP to implement inter process communication		8/1/24	
			~~5/2/24~~	
2.	WAP to implement Group communication		5/2/24	
3.	WAP to simulate Clock synchronization (Lamport's Logical Clock)		4/3/24	22/4/24
4.	WAP to simulate Election algorithm (Bully Algorithm)		11/3/24	
5.	WAP to simulate Mutual Exclusion (Raymond's Tree-Based Algorithm)		18/3/24	
6.	WAP to simulate Deadlock management in Distributed systems (Banker's Algorithm)		12/4/24	
7.	WAP to simulate Load Balancing Algorithm		17/4/24	
8.	Distributed Shared memory		22/4/24	
9.	Distributed File System		25/4/24	
10.	Case Study: COBRA, Android, stack		4/4/24	

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment 1

Aim: Write a program to implement interprocess communication

Theory:

Interprocess communication is a fundamental concept ~~to~~ in distributed computing, allowing different processes to communicate and synchronize with each other.

- Shared Memory:
Shared memory is one of the fastest IPC mechanism as it allows processes to share a portion of memory. Processes can read from and write to this shared memory region.
- Message Passing:
Message Passing involves processes communicating by sending and receiving messages. Synchronous message passing requires the sender to block until the receiver ~~ack~~ acknowledges receipt
- Sockets
Sockets are a network-based IPC mechanism that allows processes to communicate over a network. They provide reliable, bidirectional communication between processes running on different hosts.
- Pipes:
Pipes are simple form of IPC where data flows

in one direction between two processes. In Unix like operating system, pipes are implemented as file descriptors and can be either anonymous or named

- Remote Procedural Calls:
RPC allows a process to execute procedures or function in another process address space, as if they were local.

Conclusion:

Thus, we have understood the interprocess communication and successfully implemented it.

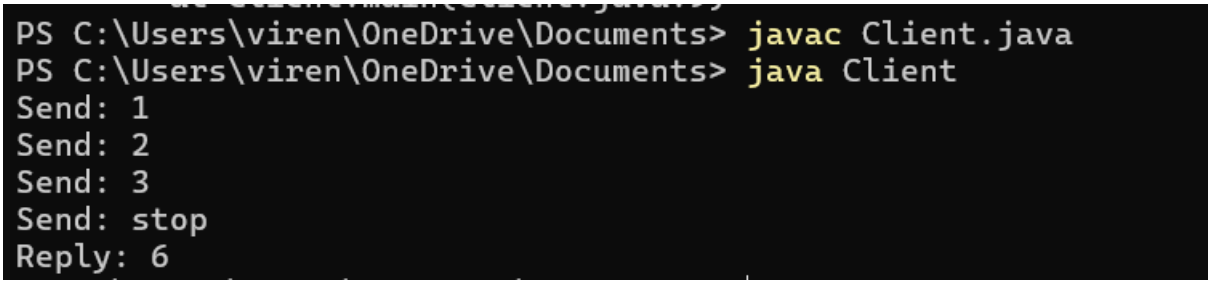
Client.java

```
import java.io.*;
import java.util.*;
import java.net.*;
public class Client
{
public static void main(String args[])throws Exception
{
String send="",r="";
Socket MyClient = new Socket("192.168.29.4",25);
DataInputStream din=new DataInputStream(MyClient.getInputStream());
DataOutputStream dout = new DataOutputStream(MyClient.getOutputStream());
Scanner sc = new Scanner(System.in);
while(!send.equals("stop")){
System.out.print("Send: ");
send = sc.nextLine();
dout.writeUTF(send);
}
dout.flush();
r=din.readUTF();
System.out.println("Reply: "+ r);
dout.close();
din.close();
MyClient.close();
}
}
```

Server.java

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

public class Server {
public static void main(String args[]) throws Exception{

ServerSocket MyServer = new ServerSocket(25);
Socket ss = MyServer.accept();
DataInputStream din =new DataInputStream(ss.getInputStream());
DataOutputStream dout=new DataOutputStream(ss.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
Server server = new Server();
String str="",str2="";
int sum = 0;
while(!str.equals("stop")){
```

```
str=din.readUTF();
if(str.equals("stop"))
break;
sum = sum + Integer.parseInt(str);
}
dout.writeUTF(Integer.toString(sum));
dout.flush();
din.close();
ss.close();
MyServer.close();
}
}
```



Experiment 2

Aim: Write a program to implement Group communication

Theory:

Group communication, also known as multicast communication, involves sending messages from one sender to multiple receiver simultaneously. This is a fundamental concept in distributed systems, where processes or nodes need to collaborate or share information within a group.

- Group Membership:
Group communication starts with defining and managing group membership. This involves maintaining a list of group members, identifying their addresses, and handling dynamic changes such as joining, leaving or failing nodes.
- Message Distribution:
Once the group membership is established, the sender needs to distribute messages to all group members efficiently. One-to-many communication methods like multicast or broadcast are used to deliver messages to multiple receipients simultaneously.

• Message Ordering:
Ensuring message ordering is essential, especially in scenarios where the order of messages is critical for correct system behaviour.
Total order multicast ensures that all group members receive messages in the same order.
Implementations often use vector clocks or timestamps to establish message causality and order.

Conclusion:

Thus, we have successfully implemented group communication.

22/1/24

Client.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;
public class GroupChatClient {
public static void main(String[] args) throws IOException {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter your name: ");
String clientName = scanner.nextLine();
Socket socket = new Socket("localhost", 5000);
try {
PrintWriter writer = new PrintWriter(socket.getOutputStream(),
true);
writer.println(clientName);
Thread readerThread = new Thread(new ClientReader(socket));
readerThread.start();
while (true) {
String message = scanner.nextLine();
writer.println(message);
if (message.equals("exit")) {
break;
}
}
} finally {
socket.close();
}
}
private static class ClientReader implements Runnable {
private Socket socket;
public ClientReader(Socket socket) {
this.socket = socket;
}
@Override
public void run() {
try {
Scanner scanner = new Scanner(socket.getInputStream());
while (scanner.hasNextLine()) {
String message = scanner.nextLine();
System.out.println(message);
}
} catch (IOException e) {
e.printStackTrace();
} finally {
```

```
System.out.println("Disconnected from the server.");
}
}
}
}
```

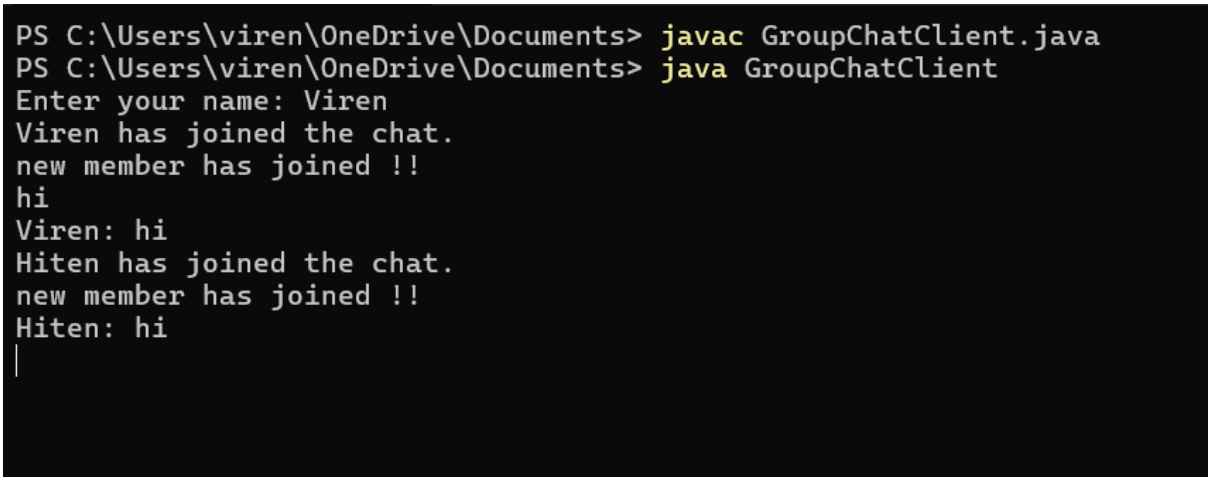
Server.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;
import java.util.concurrent.CopyOnWriteArraySet;
public class GroupChatServer {
private static Set<PrintWriter> writers = new CopyOnWriteArraySet<>();
private static Set<String> connectedClients = new HashSet<>();
public static void main(String[] args) throws IOException {
ServerSocket serverSocket = new ServerSocket(5000);
System.out.println("Group Chat Server is running...");
// Start a thread to accept messages from the server console
Thread serverMessageThread = new Thread(new
ServerMessageHandler());
serverMessageThread.start();
try {
while (true) {
Socket clientSocket = serverSocket.accept();
System.out.println("New client connected: " +
clientSocket);
Scanner clientScanner = new
Scanner(clientSocket.getInputStream());
String clientName = clientScanner.nextLine();
synchronized (writers) {
writers.add(new
PrintWriter(clientSocket.getOutputStream(), true));
connectedClients.add(clientName);
}
broadcastFromServer(clientName + " has joined the chat.");
broadcastFromServer("new member has joined !!");
Thread clientHandler = new Thread(new
ClientHandler(clientSocket, clientName));
```

```
clientHandler.start();
}
} finally {
serverSocket.close();
}
}
private static class ClientHandler implements Runnable {
private Socket clientSocket;
private String clientName;
public ClientHandler(Socket clientSocket, String clientName) {
this.clientSocket = clientSocket;
this.clientName = clientName;
}

@Override
public void run() {
try {
Scanner scanner = new
Scanner(clientSocket.getInputStream());
while (scanner.hasNextLine()) {
String message = scanner.nextLine();
if (message.equals("exit")) {
break;
} else if (message.startsWith("/whisper")) {
handleWhisper(message);
} else {
broadcast(clientName + ": " + message);
}
}
} catch (IOException e) {
e.printStackTrace();
} finally {
try {
clientSocket.close();
} catch (IOException e) {
e.printStackTrace();
}
synchronized (writers) {
writers.removeIf(writer ->
writer.equals(clientSocket));
connectedClients.remove(clientName);
}
broadcastFromServer(clientName + " has left the chat.");
}
```

```
}
private void broadcast(String message) {
synchronized (writers) {
for (PrintWriter writer : writers) {
writer.println(message);
}
}
}
private void handleWhisper(String message) {
String[] parts = message.split(" ", 3);
if (parts.length == 3) {
String recipient = parts[1].trim();
String whisperMessage = parts[2].trim();
synchronized (writers) {
if (connectedClients.contains(recipient)) {
for (PrintWriter writer : writers) {
if (writer.toString().equals(recipient)) {
writer.println("(whisper from " +
clientName + "): " + whisperMessage);
break;
}
}
} else {
writers.stream()
.filter(writer ->
writer.toString().equals(clientName))
.findFirst()

.ifPresent(writer ->

writer.println("Recipient '" + recipient + "' is not connected."));

}
}
} else {
writers.stream()
.filter(writer ->
writer.toString().equals(clientName))
.findFirst()

.ifPresent(writer -> writer.println("Invalid whisper command. Usage: /whisper recipient
message"));
}
}
```

```
}
private static class ServerMessageHandler implements Runnable {
@Override
public void run() {
try {
BufferedReader consoleReader = new BufferedReader(new
InputStreamReader(System.in));
while (true) {
String serverMessage = consoleReader.readLine();
broadcastFromServer("[host]: " + serverMessage);
}
} catch (IOException e) {
e.printStackTrace();
}
}
}
public static void broadcastFromServer(String message) {
synchronized (writers) {
for (PrintWriter writer : writers) {
writer.println(message);
}
}
}
}
```



```
PS C:\Users\viren\OneDrive\Documents> javac GroupChatClient2.java
PS C:\Users\viren\OneDrive\Documents> java GroupChatClient2
Enter your name: Hiten
Hiten has joined the chat.
new member has joined !!
hi
Hiten: hi
```

```
PS C:\Users\viren\OneDrive\Documents> javac GroupChatServer.java
PS C:\Users\viren\OneDrive\Documents> java GroupChatServer
Group Chat Server is running...
New client connected: Socket[addr=/127.0.0.1,port=51714,localport=5000]
New client connected: Socket[addr=/127.0.0.1,port=51739,localport=5000]
```

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment 3

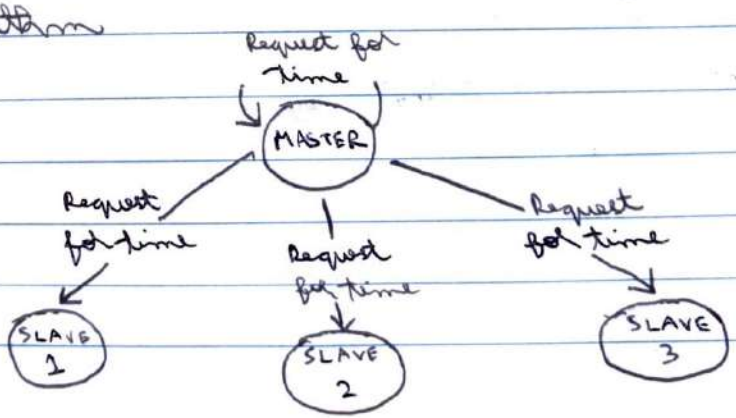
Aim: Write a program to simulate clock synchronization

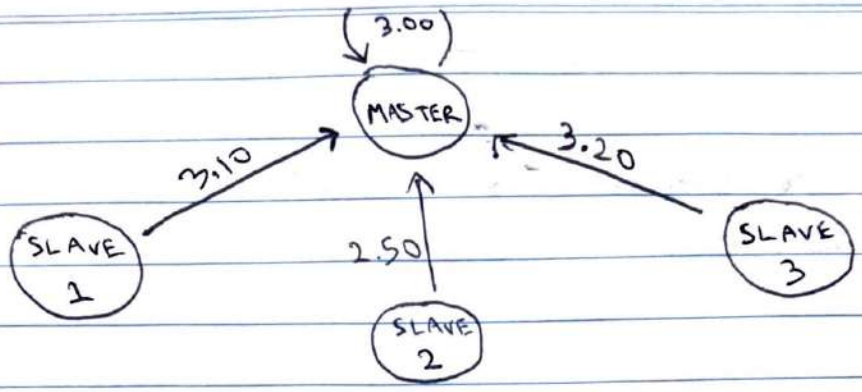
Theory:

Clock synchronization in distributed system aims to establish a reference for time across nodes. 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.

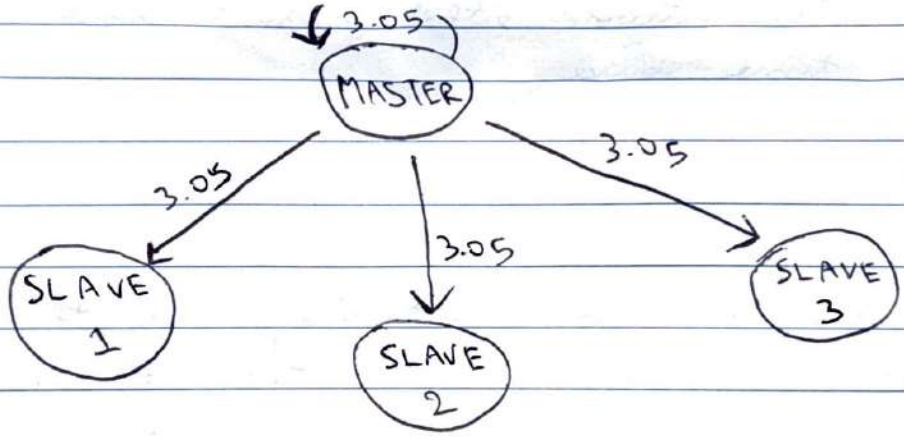
Algorithm:

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 node acts as slaves.
2) Master node periodically pings slave nodes and fetches clock time at them using Cristian's Algorithm





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.



Conclusion:

Thus, we have successfully implemented clock synchronization.

```
import java.util.Scanner;
class LamportClock {
private int time;
public LamportClock(int initialTime) {
this.time = initialTime;
}
public synchronized int getTime() {
return time;
}
public synchronized void tick() {
time++;
}
public synchronized void sendAction() {
time++;
}
public synchronized void receiveAction(int sentTime) {
time = Math.max(time, sentTime) + 1;
}
}
class Process implements Runnable {
private int processId;
private LamportClock clock;
private Scanner scanner;
public Process(int processId, LamportClock clock) {
this.processId = processId;
this.clock = clock;
this.scanner = new Scanner(System.in);
}
@Override
public void run() {
System.out.println("Process " + processId + " initial time: " + clock.getTime());
while (true) {
System.out.println("Process " + processId + ": Enter 'send' to send packet or 'exit' to quit:");
String input = scanner.nextLine();
if (input.equals("exit")) {
break;
} else if (input.equals("send")) {
sendMessage();
} else {
System.out.println("Invalid input. Please enter 'send' or 'exit'.");
}
}
}
private synchronized void sendMessage() {
```

```
int sentTime = clock.getTime();
clock.sendAction();

System.out.println("Process " + processId + " sent a packet at time " + sentTime);
// Simulating packet transmission time
try {
Thread.sleep(1000); // Sleep for 1 second
} catch (InterruptedException e) {
e.printStackTrace();
}
// Receive the packet
int receivedTime = clock.getTime();
clock.receiveAction(sentTime);
System.out.println("Process " + processId + " received a packet at time " + receivedTime);
}
}
public class LamportClockSimulation {
public static void main(String[] args) {
LamportClock clock1 = new LamportClock(0); // Initial time for process 1
LamportClock clock2 = new LamportClock(5); // Initial time for process 2
Thread process1 = new Thread(new Process(1, clock1));
Thread process2 = new Thread(new Process(2, clock2));
process1.start();
process2.start();
}
}
```

```
PS C:\Users\viren\OneDrive\Documents> java LamportClockSimulation
Process 2 initial time: 5
Process 1 initial time: 0
Process 2: Enter 'send' to send packet or 'exit' to quit:
Process 1: Enter 'send' to send packet or 'exit' to quit:
send
send
Process 1 sent a packet at time 0
Process 2 sent a packet at time 5
Process 1 received a packet at time 1
Process 2 received a packet at time 6
Process 2: Enter 'send' to send packet or 'exit' to quit:
Process 1: Enter 'send' to send packet or 'exit' to quit:
send
send
Process 2 sent a packet at time 7
Process 1 sent a packet at time 2
Process 1 received a packet at time 3
Process 2 received a packet at time 8
Process 1: Enter 'send' to send packet or 'exit' to quit:
Process 2: Enter 'send' to send packet or 'exit' to quit:
```

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment 4

Aim: Write a program to simulate Election Algorithm (Bully Algorithm)

Theory:

The Bully algorithm is a type of Election algorithm which is mainly used for choosing a coordinate. Election algorithms select a single process from the processes that acts as a coordinator. There are three types of messages that process exchange with each other in bully algorithm.

Election Message: Sent to announce election

OK Message: Responds to the Election message

Coordinator: sent by winner of the election to announce the new coordinator

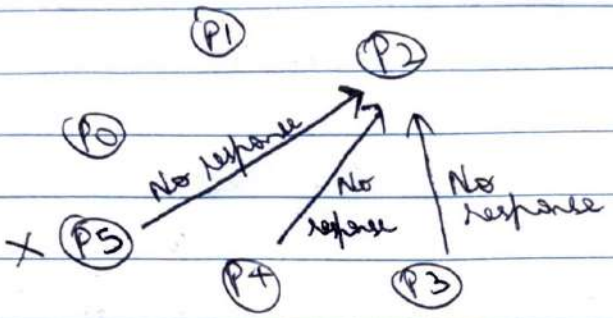
Steps:

Step 1: Suppose process P2 sends a message to coordinator P5 & P5 doesn't respond in desired time T

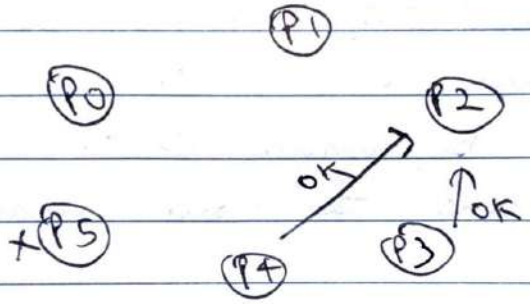
Step 2: Then process P2, sends an election message to all processes with process ID greater than P2 and awaits a response from the process.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

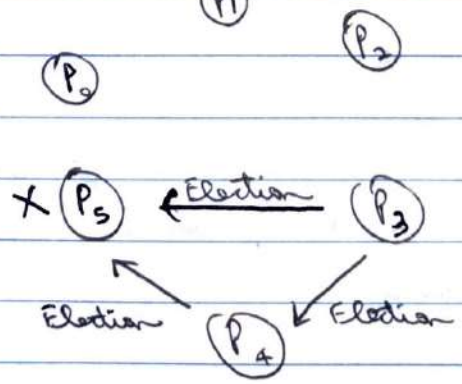
Step 3: If none responds, P2 wins the election & becomes the coordinator.



Step 4: If any of the processes with Process ID higher than 2 responds with OK P2's job is done and that process will take over.

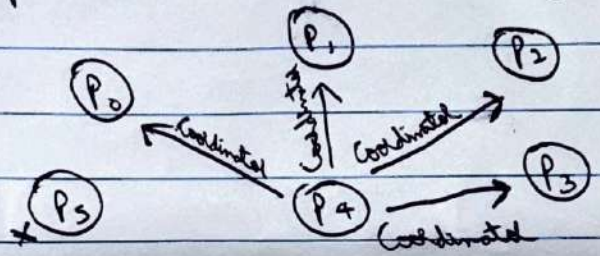


Step 5: It then restarts and initiates an election message



Step 6: Process P_4 responds to P_3 with an OK message to confirm its alive status and process P_4 figures out that process 5 has crashed and the new process with highest ID is process 4

Step 7: The process that receives an election messag sends a coordinator message if it is the process with the highest ID.



Step 8: If a process which was previously down (ie P5) comes back, it holds an election.

Conclusion:

Thus, we have understood and successfully implemented Bully's algorithm.

22/4/24

```
import java.util.Scanner;
class Pro {
int id;
boolean act;
Pro(int id) {
this.id = id;
this.act = true;
}
}
public class Bully {
int TotalProcess;
Pro[] process;
public static void main(String[] args) {
Bully object = new Bully();
Scanner scanner = new Scanner(System.in);
System.out.print("Total number of processes: ");
int numProcesses = scanner.nextInt();
object.initialize(numProcesses);
object.Election();
scanner.close();
}
public void initialize(int j) {
System.out.println("No of processes " + j);
this.TotalProcess = j;
this.process = new Pro[this.TotalProcess];
for (int i = 0; i < this.TotalProcess; i++) {
this.process[i] = new Pro(i);
}
}
public void Election() {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the process number to fail: ");
int process_to_fail = scanner.nextInt();
if (process_to_fail < 0 || process_to_fail >= this.TotalProcess) {
System.out.println("Invalid process number. Please enter a valid process number.");
return;
}
System.out.println("Process no " + this.process[process_to_fail].id + " fails");
this.process[process_to_fail].act = false;
System.out.print("Enter the process initiating the election: ");
int initializedProcess = scanner.nextInt();
System.out.println("Election Initiated by " + initializedProcess);
int initBy = initializedProcess;
int i = -1;
```

```
for(i = initBy; i < TotalProcess - 1; i++){
if(this.process[i].act){
System.out.println("Process " + i + " passes Election(" + i + ")" + " to " +(i + 1));
}else{
System.out.println("Process " + (i-1) + " passes Election(" + (i - 1) + ")" + " to " +(i + 1));
}
}
int coordinatorID = -1;
if(this.process[i].act) {
coordinatorID = i;
}else{
coordinatorID = i - 1;
}
System.out.println("Process " + coordinatorID + " is the process with max ID");
for(int j = 0; j< coordinatorID; j++){
if(this.process[j].act)
System.out.println("Process " + coordinatorID + " passes Coordinator(" + coordinatorID + ")
message to process " + this.process[j].id);
}
scanner.close();
}
}
```

```
PS C:\Users\viren\OneDrive\Documents> javac Bully.java
PS C:\Users\viren\OneDrive\Documents> java Bully
Total number of processes: 9
No of processes 9
Enter the process number to fail:
3
Process no 3 fails
Enter the process initiating the election: 5
Election Initiated by 5
Process 5 passes Election(5) to 6
Process 6 passes Election(6) to 7
Process 7 passes Election(7) to 8
Process 8 is the process with max ID
Process 8 passes Coordinator(8) message to process 0
Process 8 passes Coordinator(8) message to process 1
Process 8 passes Coordinator(8) message to process 2
Process 8 passes Coordinator(8) message to process 4
Process 8 passes Coordinator(8) message to process 5
Process 8 passes Coordinator(8) message to process 6
Process 8 passes Coordinator(8) message to process 7
```

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment 5

Aim: Write a program to simulate Mutual Exclusion (Raymonds Tree-Based Algorithm)

Theory: Raymond's Tree based algorithm is a distributed algorithm used to achieve mutual exclusion among processes in a distributed algorithm. It allows processes to enter critical section in a mutualy exclusive manner without relying on a centralized coordinator.

1. Tree structure: The algorithm relies on a logical tree structure where each process is represented as a node in the tree.

2. Node State: Each process maintains local state information required to participate in the algorithm. Key components of the local state include the process's unique identifier, its position in the tree and flags.

3. Requesting critical section entry: When a process desires to enter the critical section, it initiates the entry request by sending a message to its parent node in the tree.

4. Propagation of Requests: Upon receiving a request message from a child node, a process may propagate the request further up the tree by forwarding the request to its parent node. This propagation continues until the request

reaches a process that is currently executing the critical section.

5) Acknowledgment:
As the request propagates up the tree, each intermediate node acknowledges the receipt of the request. Acknowledgment serves to request has been successfully transmitted.

6. Granting Permission: When the request reaches the root of the tree, the root node grants permission to enter the critical section which propagates this permission back down the tree, following reverse path of request.

7. Entering critical section: Upon receiving permission from the root node, process enters critical section. While inside it, process performs its exclusive operations, ensuring mutual exclusion.

8. Exiting critical section: After completing its critical section operations, the process releases the critical section & notifies its parent node that it's no longer interested in entering.

9. Release of Permission: As the notifications propagates up the tree, each intermediate node releases any permissions it holds for the corresponding child process.

10. Handling concurrent Requests: This algorithm ensures fairness by prioritizing the requests based on their positions in the tree. If multiple requests arrive simultaneously at an intermediate node, the node may grant permission to the request with highest priority, based on predefined criteria

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.Scanner;
public class RaymondTree {
static List<Node> list = new ArrayList<>();
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
int n = 8;
Node n1 = new Node(1, 1);
Node n2 = new Node(2, 1);
Node n3 = new Node(3, 1);
Node n4 = new Node(4, 2);
Node n5 = new Node(5, 2);
Node n6 = new Node(6, 3);
Node n7 = new Node(7, 3);
Node n8 = new Node(8, 3);
list.add(n1);
list.add(n2);
list.add(n3);
list.add(n4);
list.add(n5);
list.add(n6);
list.add(n7);
list.add(n8);
while (true) {
System.out.println("Enter the ID of the node requesting access to the critical section (1-8), or
'exit' to quit:");
String input = scanner.nextLine();
if (input.equalsIgnoreCase("exit")) {
break;
}
int nodeID;
try {
nodeID = Integer.parseInt(input);
if (nodeID < 1 || nodeID > 8) {
System.out.println("Invalid node ID. Please enter a number between 1 and 8.");
continue;
}
} catch (NumberFormatException e) {
System.out.println("Invalid input. Please enter a valid node ID or 'exit'.");
continue;
}
```

```
List<Integer> req = new LinkedList<>();
req.add(nodeID);
System.out.println("Initially:");
printFunction();
for (int a : req) {
System.out.println("--------- Process " + a + " ----------- ");
request(a, a);
}
}

scanner.close();
}
private static void request(int holder, int req) {
Node n = list.get(holder - 1);
n.q.add(req);
System.out.println("Queue of " + n.id + ": " + n.q);
if (n.holder != n.id)
request(n.holder, n.id);
else
giveToken(n.id);
}
private static void giveToken(int nid) {
Node n = list.get(nid - 1);
int next = n.q.remove();
n.holder = next;
if (next != nid)
giveToken(next);
else
printFunction();
}
private static void printFunction() {
for (Node n : list) {
System.out.println("id: " + n.id + " Holder: " + n.holder);
}
}
}
class Node {
int id;
int holder;
Queue<Integer> q;
Node(int id, int holder) {
this.id = id;
this.holder = holder;
this.q = new LinkedList<>();
```

```
}
}
```

```
PS C:\Users\viren\OneDrive\Documents> javac RaymondTree.java
PS C:\Users\viren\OneDrive\Documents> java RaymondTree
Enter the ID of the node requesting access to the critical section (1-8), or 'exit' to quit:
3
Initially:
id: 1 Holder: 1
id: 2 Holder: 1
id: 3 Holder: 1
id: 4 Holder: 2
id: 5 Holder: 2
id: 6 Holder: 3
id: 7 Holder: 3
id: 8 Holder: 3
--------- Process 3 ----------
Queue of 3: [3]
Queue of 1: [3]
id: 1 Holder: 3
id: 2 Holder: 1
id: 3 Holder: 3
id: 4 Holder: 2
id: 5 Holder: 2
id: 6 Holder: 3
id: 7 Holder: 3
id: 8 Holder: 3
Enter the ID of the node requesting access to the critical section (1-8), or 'exit' to quit:
```

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment 6

Aim: Write a program to simulate deadlock management in Distributed Systems (Banker's Algorithm)

Theory: Distributed systems have a banker's algorithm which serves as a deadlock avoidance algorithm. Banker's algorithm depicts the resource allocation strategy which can help in determining the availability of resources. To avoid deadlock in a system, Banker's algorithm includes the below two algorithms:

- Resource Request Algorithm
- Safety Algorithm

Safety Algorithm: This algorithm is used to determine and calculate the safety of the system. To determine the safety of the ~~algorithm~~ system, it's important to understand: Work, Finish[i], and Need[i] w.r.t. the resources

Step 1: Set work as Available.

Work = available -① Finish[i] = false -②

Step 2: Check the availability of resources & continue the loop til Finish[i] is valued as false.

Need[i] <= Work -③

Finish[i] == false -④

Work = Work + Allocation(i) -⑤

Step 3: When Finish[i] is valued as true we can consider the system to be safe.

Finish[i] == true -⑥

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Resource Request Algorithm:

The resource request algorithm as the name suggests is used to determine and calculate the system, behaviour when process makes request to resources.

To determine the resource request of the system, it's important to understand: Allocation[i], Resource[i], Process[i] w.r.t. the resources.

Step 1:

Request(i) <= Need - (1)

Step 2:

Request(i) <= Available - (2)

Step 3:

Available = Available - Request - (3)

Allocation(i) = Allocation(i) + Request(i) - (4)

Need(i) = Need(i) - Request(i) - (5)

22 values

```
import java.util.Scanner;
public class BankersAlgorithm {
private int[][] allocation;
private int[][] max;
private int[] available;
private int[][] need;
private int numProcesses;
private int numResources;
public BankersAlgorithm(int numProcesses, int numResources) {
this.numProcesses = numProcesses;
this.numResources = numResources;
allocation = new int[numProcesses][numResources];
max = new int[numProcesses][numResources];
available = new int[numResources];
need = new int[numProcesses][numResources];
}
public void getInput() {
Scanner scanner = new Scanner(System.in);
// Input allocation matrix
System.out.println("Enter Allocation Matrix:");
for (int i = 0; i < numProcesses; i++) {
for (int j = 0; j < numResources; j++) {
allocation[i][j] = scanner.nextInt();
}
}
// Input max matrix
System.out.println("Enter Max Matrix:");
for (int i = 0; i < numProcesses; i++) {
for (int j = 0; j < numResources; j++) {
max[i][j] = scanner.nextInt();
need[i][j] = max[i][j] - allocation[i][j];
}
}
// Input available resources
System.out.println("Enter Available Resources:");
for (int i = 0; i < numResources; i++) {
available[i] = scanner.nextInt();
}
}
public boolean isSafe(int process) {
int[] tempAvailable = available.clone();
int[][] tempAllocation = new int[numProcesses][numResources];
int[][] tempNeed = new int[numProcesses][numResources];
// Check if the system is still in safe state after resource allocation
```

```
boolean[] finished = new boolean[numProcesses];
for (int i = 0; i < numProcesses; i++) {
finished[i] = false;
}
// Try to allocate the resources for the current process

for (int i = 0; i < numResources; i++) {
tempAvailable[i] -= need[process][i];
tempAllocation[process][i] += need[process][i];
tempNeed[process][i] -= need[process][i];
}
int count = 0;
while (count < numProcesses) {
boolean found = false;
for (int i = 0; i < numProcesses; i++) {
if (!finished[i]) {
boolean safe = true;
for (int j = 0; j < numResources; j++) {
if (tempNeed[i][j] > tempAvailable[j]) {
safe = false;
break;
}
}

if (safe) {

finished[i] = true;
count++;
found = true;
for (int j = 0; j < numResources; j++) {
tempAvailable[j] += tempAllocation[i][j];
}
}
}
}
if (!found) {
break;
}
}
return count == numProcesses;
}
public void grantResources() {
for (int i = 0; i < numProcesses; i++) {
if (isSafe(i)) {
```

```
for (int j = 0; j < numResources; j++) {
available[j] += allocation[i][j];
allocation[i][j] = 0;
need[i][j] = 0;
}
System.out.println("Resources granted for process " + (i + 1) + " successfully.");
System.out.println("Available resources after process " + (i + 1) + ":");
for (int k = 0; k < numResources; k++) {
System.out.print(available[k] + " ");
}
System.out.println();
} else {
System.out.println("Unsafe state, resources cannot be granted for process " + (i + 1) + ".");
}
}
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);

// Input number of processes and resources
System.out.println("Enter number of processes:");
int numProcesses = scanner.nextInt();
System.out.println("Enter number of resources:");
int numResources = scanner.nextInt();
BankersAlgorithm bankers = new BankersAlgorithm(numProcesses,
numResources);
bankers.getInput();
bankers.grantResources();
}
}
```

```
PS C:\Users\viren\OneDrive\Documents> java BankersAlgorithm
Enter number of processes:
3
Enter number of resources:
3
Enter Allocation Matrix:
1 0 1
1 1 1
1 0 1
Enter Max Matrix:
2 2 2
1 2 2
3 2 2
Enter Available Resources:
1 0 1
Resources granted for process 1 successfully.
Available resources after process 1:
2 0 2
Resources granted for process 2 successfully.
Available resources after process 2:
3 1 3
Resources granted for process 3 successfully.
Available resources after process 3:
4 1 4
```

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

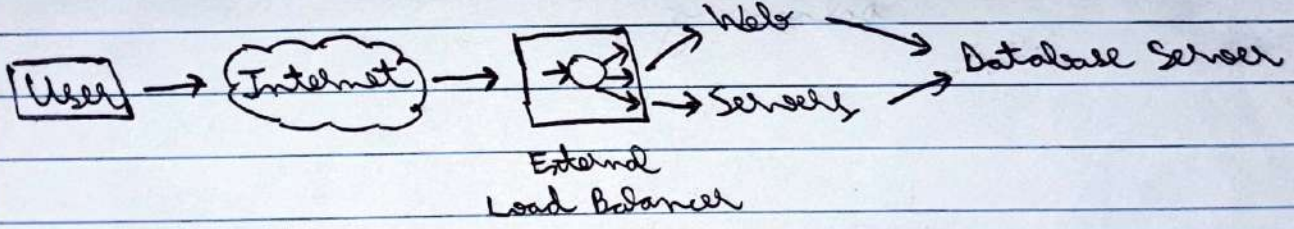
Experiment 7

Aim: Write a program to simulate Load Balancing algorithm

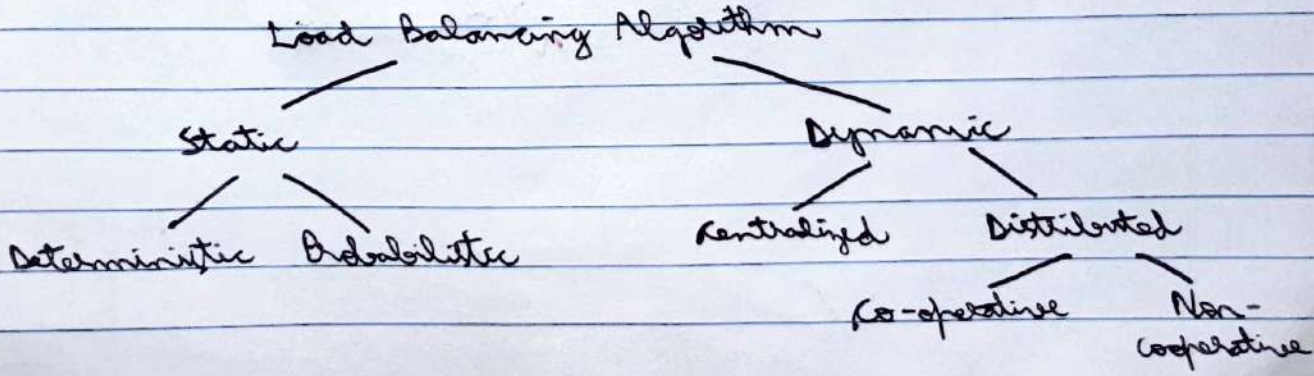
Theory:

The function of a load balancer is to distribute network or application traffic among several available servers by acting as a reverse proxy. The method of distributing load units around the organisation that is connected to the distributed system is called load ~~balancing~~ adjusting. Load balancer accomplishes following tasks:

- Efficiently divides client requests or network strain among several servers
- Provides flexibility to add or remove servers as needed



Classification of Load Balancing Algorithms:



- Static - Designed for system with low fluctuation in incoming load
- Dynamic - Designed for system with high fluctuation in incoming load
- Deterministic - Use information about node properties and characteristics of processes to be scheduled
- Probabilistic - Use information of static attributes of the system to formulate simple process placement rules
- Centralized - Collects information to server and makes assignment decisions
- Distributed - Contains entities to make decisions on a pre-defined set of nodes

Conclusion: Successfully implemented load balancing.

```
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;
}
}
}
}
```

```
PS C:\Users\viren\OneDrive\Documents> javac LoadBalancer.java
PS C:\Users\viren\OneDrive\Documents> java LoadBalancer
Enter the number of servers and Processes: 5 10
Server A has 2 Processes
Server B has 2 Processes
Server C has 2 Processes
Server D has 2 Processes
Server E has 2 Processes
1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 1
How many more servers?: 4
Server A has 2 Processes
Server B has 1 Processes
Server C has 1 Processes
Server D has 1 Processes
Server E has 1 Processes
Server F has 1 Processes
Server G has 1 Processes
Server H has 1 Processes
Server I has 1 Processes
1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 2
How many servers to remove?: 3
Server A has 2 Processes
Server B has 2 Processes
Server C has 2 Processes
Server D has 2 Processes
Server E has 1 Processes
Server F has 1 Processes
1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 3
How many more Processes?: 1
Server A has 2 Processes
Server B has 2 Processes
Server C has 2 Processes
Server D has 2 Processes
Server E has 2 Processes
Server F has 1 Processes
1.Add Servers 2.Remove Servers 3.Add Processes 4.Remove Processes 5.Exit: 5
```

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

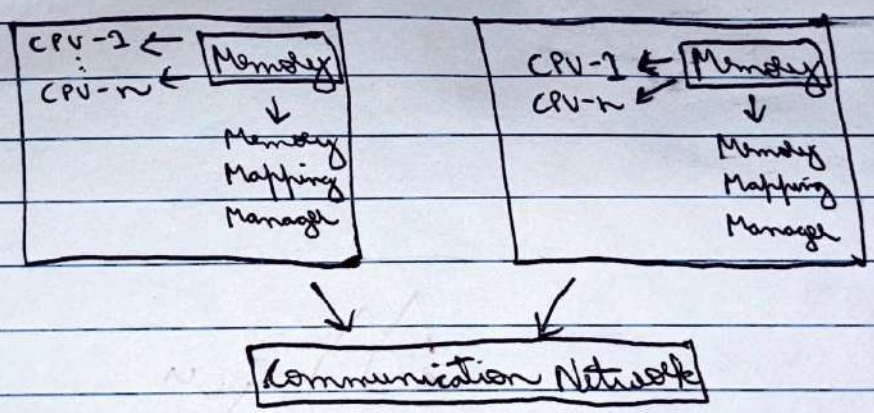
Experiment 8

Aim- Distributed Shared Memory

Theory-

DSM is a mechanism that manage memory across multiple nodes and makes interprocess communications transparent to end-users. DSM is a mechanism of allowing user processes to access shared data without interprocess communications. All nodes share the virtual address space provided by the shared memory model. The data moves between the main memories of different nodes.

DSM (Exists only virtually)



Advantages of DSM-

- Simpler abstraction - Programmer need not be concern about data movement. As the address space is the same it is easier to implement than RPC

- Locality of Data - Data moved in large blocks ie. data near to the current memory location that is being fetched, may be needed in future so it will also be fetched

- Large memory space - It provides large virtual memory space, the total memory size is the sum of the memory size of all nodes.

- Better performance - DSM improves performance & efficiency by speeding up access to data.

Conclusion: Successfully implemented Distributed Shared Memory.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment 9

Aim- Distributed File System

Theory-

A distributed file system (DFS) is a file system that is distributed on multiple file servers or locations. It allows programs to access or store isolated files as they do with the local ones, allowing programmers to access file from any network or computer.

Features of DFS are:

- Transparency:-
  - Structural Transparency: Clients should not know the no. of locations of file servers & the storage devices
  - Access Transparency: Both local & remote files should be accessible in the same way
  - Naming Transparency: The name of the file should give no hint as to the location of the file. The name must not be changed when moving from one node to another.
  - Replication Transparency: If a file is replicated on multiple nodes, both the existence of multiple copies & their locations should be hidden from the clients.
- User Mobility :- Users should be able to access file from anywhere

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

- Performance: Measured as the average amount of time needed to satisfy client ~~secretary~~ requests.

- Simplicity and ease of use: User interface to the file system should be simple and number of commands should be as small as possible.

- Scalability: Growth of nodes and users should not seriously disrupt service

Conclusion:-

Successfully implemented.

22/1/24

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Experiment 10

Aim: Case study Cobra, Android Stack

Theory:

1) COBRA (Common Object Request Broker Architecture)

- COBRA could be a specification of a regular design for middleware. It's a client server development model. Using COBRA implementation, a shopper will transparently invoke a may on a server object, which may sit a similar machine or across a network.

COBRA Reference Model:

Cobra Reference Model known as object management design (OMA) is shown below. the OMA is itself a specification, that ~~defies~~ defines a broad variety of services for building distributed client server applications.

Several applications may expect to search out in every middleware product like Cobra.