IPC Folder:

Step 1: Make Client.java

import java.io.\*;

import java.net.\*;

public class Client {

    public static void main(String[] args) {

        try {

            Socket socket = new Socket("localhost", 12345);

            System.out.println("Connected to server.");

            BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            String message;

            while (true) {

                System.out.print("Client: ");

                message = userInput.readLine();

                if (message.equals("exit")) break;

                out.println(message);

                System.out.println("Server: " + in.readLine());

            }

            userInput.close();

            in.close();

            out.close();

            socket.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

Step 2: Make Server.java

import java.io.\*;

import java.net.\*;

public class Server {

    public static void main(String[] args) {

        try {

            ServerSocket serverSocket = new ServerSocket(12345);

            System.out.println("Server started. Waiting for a client...");

            Socket clientSocket = serverSocket.accept();

            System.out.println("Client connected.");

            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

            String message;

            while ((message = in.readLine()) != null) {

                System.out.println("Client: " + message);

                out.println("Server received: " + message);

            }

            in.close();

            out.close();

            clientSocket.close();

            serverSocket.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

Step 3: javac Client.java

Javac Server.java

Step 4: java Server

Java Client

Group communication:

Step 1: Make MulticastReceiver.java

import java.io.\*;

import java.net.\*;

public class MulticastReceiver {

    public static void main(String[] args) {

        try {

            InetAddress group = InetAddress.getByName("239.0.0.1");

            MulticastSocket socket = new MulticastSocket(12345);

            socket.joinGroup(group);

            byte[] buffer = new byte[1024];

            DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

            while (true) {

                socket.receive(packet);

                String message = new String(packet.getData(), 0, packet.getLength());

                System.out.println("Receiver: " + message);

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

Step 2: Make MulticastSender.java

import java.io.\*;

import java.net.\*;

public class MulticastSender {

    public static void main(String[] args) {

        try {

            InetAddress group = InetAddress.getByName("239.0.0.1");

            MulticastSocket socket = new MulticastSocket();

            BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

            String message;

            while (true) {

                System.out.print("Sender: ");

                message = userInput.readLine();

                if (message.equals("exit")) break;

                byte[] buffer = message.getBytes();

                DatagramPacket packet = new DatagramPacket(buffer, buffer.length, group, 12345);

                socket.send(packet);

            }

            userInput.close();

            socket.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

Step 3: Compile both

Step 4: First run receiver , then sender….(exit to end)

BullyAlgorithm.java:

// import required classes and packages

import java.util.Scanner;

// create process class for creating a process having id and status

class Process{

    // declare variables

    public int id;

    public String status;

    // initialize variables using constructor

    public Process(int id){

        this.id = id;

        this.status = "active";

    }

}

// create class BullyAlgoExample2 for understanding the concept of Bully algorithm

public class BullyAlgorithm {

    // initialize variables and array

    Scanner sc;

    Process[] processes;

    int n;

    // initialize Scanner class object in constructor

    public BullyAlgorithm(){

        sc= new Scanner(System.in);

    }

    // create ring() method for initializing the ring

    public void ring(){

        // get input from the user for processes

        System.out.println("Enter total number of processes of Processes");

        n = sc.nextInt();

        // initialize processes array

        processes = new Process[n];

        for(int i = 0; i< n; i++){

            processes[i]= new Process(i);

        }

    }

    // create election() method for electing process

    public void performElection(){

        // we use the sleep() method to stop the execution of the current thread

        try {

            Thread.sleep(1000);

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

        // show failed process

        System.out.println("Process having id "+processes[getMaxValue()].id+" fails");

        // change status to Inactive of the failed process

        processes[getMaxValue()].status = "Inactive";

        // declare and initialize variables

        int idOfInitiator = 0;

        boolean overStatus = true;

        // use while loop to repeat steps

        while(overStatus){

            boolean higherProcesses = false;

            // iterate all the processes

            for(int i = idOfInitiator + 1; i< n; i++){

                if(processes[i].status == "active"){

                    System.out.println("Process "+idOfInitiator+" Passes Election("+idOfInitiator+") message to process" +i);

                    higherProcesses = true;

                }

            }

            // check for higher process

            if(higherProcesses){

                // use for loop to again iterate processes

                for(int i = idOfInitiator + 1; i< n; i++){

                    if(processes[i].status == "active"){

                        System.out.println("Process "+i+"Passes Ok("+i+") message to process" +idOfInitiator);

                    }

                }

                // increment initiator id

                idOfInitiator++;

            }

            else{

                // get the last process from the processes that will become coordinator

                int coord = processes[getMaxValue()].id;

                // show process that becomes the coordinator

                System.out.println("Finally Process "+coord+" Becomes Coordinator");

                for(int i = coord - 1; i>= 0; i--){

                    if(processes[i].status == "active"){

                        System.out.println("Process "+coord+"Passes Coordinator("+coord+") message to process " +i);

                    }

                }

                System.out.println("End of Election");

                overStatus = false;

                break;

            }

        }

    }

    // create getMaxValue() method that returns index of max process

    public int getMaxValue(){

        int mxId = -99;

        int mxIdIndex = 0;

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

            if(processes[i].status == "active" && processes[i].id >mxId){

                mxId = processes[i].id;

                mxIdIndex = i;

            }

        }

        return mxIdIndex;

    }

    // main() method start

    public static void main(String[] args) {

        // create instance of the BullyAlgoExample2 class

        BullyAlgorithm bully = new BullyAlgorithm();

        // call ring() and performElection() method

        bully.ring();

        bully.performElection();

    }

}

RingAlgorithm.java

import java.util.\*;

class RingNode {

    private int id;

    private RingNode successor;

    public RingNode(int id) {

        this.id = id;

    }

    public void setSuccessor(RingNode successor) {

        this.successor = successor;

    }

    public RingNode getSuccessor() {

        return successor;

    }

    public int getId() {

        return id;

    }

}

public class RingAlgorithm {

    public static void main(String[] args) {

        // Create ring nodes

        RingNode node1 = new RingNode(1);

        RingNode node2 = new RingNode(2);

        RingNode node3 = new RingNode(3);

        RingNode node4 = new RingNode(4);

        // Set successors

        node1.setSuccessor(node2);

        node2.setSuccessor(node3);

        node3.setSuccessor(node4);

        node4.setSuccessor(node1);

        // Start election from node 1

        electLeader(node1);

    }

    public static void electLeader(RingNode startNode) {

        RingNode currentNode = startNode;

        RingNode leaderNode = startNode;

        do {

            System.out.println("Node " + currentNode.getId() + " sends election message to node " + currentNode.getSuccessor().getId());

            currentNode = currentNode.getSuccessor();

            if (currentNode.getId() > leaderNode.getId()) {

                leaderNode = currentNode;

            }

        } while (currentNode != startNode);

        System.out.println("Leader elected: Node " + leaderNode.getId());

    }

}

BerkeleyAlgorithm.java

import java.util.ArrayList;

import java.util.List;

class Node {

    private int id;

    private long timeOffset;

    public Node(int id) {

        this.id = id;

        this.timeOffset = 0;

    }

    public int getId() {

        return id;

    }

    public long getTimeOffset() {

        return timeOffset;

    }

    public void setTimeOffset(long timeOffset) {

        this.timeOffset = timeOffset;

    }

    // Simulating clock drift by adding an offset to the current time

    public long getCurrentTime() {

        return System.currentTimeMillis() + timeOffset;

    }

}

public class BerkeleyAlgorithm {

    public static void main(String[] args) {

        // Create nodes

        Node node1 = new Node(1);

        Node node2 = new Node(2);

        Node node3 = new Node(3);

        // Add nodes to a list

        List<Node> nodes = new ArrayList<>();

        nodes.add(node1);

        nodes.add(node2);

        nodes.add(node3);

        // Calculate average time

        long averageTime = calculateAverageTime(nodes);

        // Synchronize clocks

        synchronizeClocks(nodes, averageTime);

        // Print synchronized times

        System.out.println("Synchronized Times:");

        for (Node node : nodes) {

            System.out.println("Node " + node.getId() + ": " + node.getCurrentTime());

        }

    }

    public static long calculateAverageTime(List<Node> nodes) {

        long sum = 0;

        for (Node node : nodes) {

            sum += node.getCurrentTime();

        }

        return sum / nodes.size();

    }

    public static void synchronizeClocks(List<Node> nodes, long averageTime) {

        for (Node node : nodes) {

            long timeOffset = averageTime - node.getCurrentTime();

            node.setTimeOffset(timeOffset);

        }

    }

}

Suzuki Kasami algorithm.java

import java.util.concurrent.Semaphore;

class Process extends Thread {

    private int id;

    private int numProcesses;

    private Semaphore mutex;

    private Semaphore[] reply;

    private int[] request;

    public Process(int id, int numProcesses, Semaphore mutex, Semaphore[] reply, int[] request) {

        this.id = id;

        this.numProcesses = numProcesses;

        this.mutex = mutex;

        this.reply = reply;

        this.request = request;

    }

    public void run() {

        while (true) {

            try {

                // Non-critical section

                Thread.sleep((long) (Math.random() \* 1000));

                // Entering the critical section

                requestCS();

                enterCS();

                leaveCS();

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

        }

    }

    private void requestCS() throws InterruptedException {

        mutex.acquire();

        request[id] = 1;

        mutex.release();

        for (int j = 0; j < numProcesses; j++) {

            if (j != id) {

                reply[j].release();

            }

        }

        for (int j = 0; j < numProcesses; j++) {

            if (j != id) {

                reply[j].acquire();

            }

        }

    }

    private void enterCS() throws InterruptedException {

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

    }

    private void leaveCS() throws InterruptedException {

        mutex.acquire();

        request[id] = 0;

        for (int j = 0; j < numProcesses; j++) {

            if (request[j] == 1) {

                reply[j].release();

            }

        }

        mutex.release();

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

    }

}

public class SuzukiKasamiAlgorithm {

    public static void main(String[] args) {

        int numProcesses = 5;

        Semaphore mutex = new Semaphore(1);

        Semaphore[] reply = new Semaphore[numProcesses];

        int[] request = new int[numProcesses];

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

            reply[i] = new Semaphore(0);

            request[i] = 0;

        }

        Process[] processes = new Process[numProcesses];

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

            processes[i] = new Process(i, numProcesses, mutex, reply, request);

            processes[i].start();

        }

    }

}

RicartAgrawalaAlgorithm.java

import java.util.concurrent.Semaphore;

class Process extends Thread {

    private int id;

    private int numProcesses;

    private Semaphore mutex;

    private Semaphore[] csEntered;

    private boolean inCS;

    private int[] requestQueue;

    public Process(int id, int numProcesses, Semaphore mutex, Semaphore[] csEntered, int[] requestQueue) {

        this.id = id;

        this.numProcesses = numProcesses;

        this.mutex = mutex;

        this.csEntered = csEntered;

        this.inCS = false;

        this.requestQueue = requestQueue;

    }

    public void run() {

        while (true) {

            try {

                // Non-critical section

                Thread.sleep((long) (Math.random() \* 1000));

                // Entering the critical section

                requestCS();

                enterCS();

                leaveCS();

            } catch (InterruptedException e) {

                e.printStackTrace();

                // Handle the exception

            }

        }

    }

    private void requestCS() throws InterruptedException {

        mutex.acquire();

        requestQueue[id] = 1;

        mutex.release();

        for (int j = 0; j < numProcesses; j++) {

            if (j != id) {

                while (requestQueue[j] == 1 || (inCS && csEntered[j].availablePermits() == 0)) {

                    // Wait until it's our turn

                    Thread.sleep(10);

                }

            }

        }

    }

    private void enterCS() throws InterruptedException {

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

        inCS = true;

    }

    private void leaveCS() throws InterruptedException {

        mutex.acquire();

        requestQueue[id] = 0;

        for (int j = 0; j < numProcesses; j++) {

            if (j != id) {

                csEntered[j].release();

            }

        }

        mutex.release();

        inCS = false;

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

    }

}

public class RicartAgrawalaAlgorithm {

    public static void main(String[] args) {

        int numProcesses = 5;

        Semaphore mutex = new Semaphore(1);

        Semaphore[] csEntered = new Semaphore[numProcesses];

        int[] requestQueue = new int[numProcesses];

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

            csEntered[i] = new Semaphore(0);

            requestQueue[i] = 0;

        }

        Process[] processes = new Process[numProcesses];

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

            processes[i] = new Process(i, numProcesses, mutex, csEntered, requestQueue);

            processes[i].start();

        }

    }

}

Round robin load balancer

import java.util.ArrayList;

import java.util.List;

class Server {

    private int id;

    private int load;

    public Server(int id) {

        this.id = id;

        this.load = 0;

    }

    public int getId() {

        return id;

    }

    public int getLoad() {

        return load;

    }

    public void incrementLoad() {

        load++;

    }

}

public class RoundRobinLoadBalancer {

    private List<Server> servers;

    private int currentIndex;

    public RoundRobinLoadBalancer() {

        servers = new ArrayList<>();

        currentIndex = 0;

    }

    public void addServer(Server server) {

        servers.add(server);

    }

    public Server getNextServer() {

        Server nextServer = servers.get(currentIndex);

        currentIndex = (currentIndex + 1) % servers.size(); // Move to the next server in a circular manner

        return nextServer;

    }

    public static void main(String[] args) {

        RoundRobinLoadBalancer loadBalancer = new RoundRobinLoadBalancer();

        // Add some servers

        loadBalancer.addServer(new Server(1));

        loadBalancer.addServer(new Server(2));

        loadBalancer.addServer(new Server(3));

        // Simulate requests

        for (int i = 0; i < 10; i++) {

            Server server = loadBalancer.getNextServer();

            server.incrementLoad();

            System.out.println("Request assigned to Server " + server.getId());

        }

        // Print server loads

        System.out.println("\nServer Loads:");

        for (Server server : loadBalancer.servers) {

            System.out.println("Server " + server.getId() + ": " + server.getLoad());

        }

    }

}

RMI: