**CODE: (Bully Algorithm)**

import java.util.ArrayList;

import java.util.List;

public class BullyAlgorithm {

private int processId;

private List<Integer> processes;

public BullyAlgorithm(int processId, List<Integer> processes) {

this.processId = processId;

this.processes = processes;

}

public int run() {

boolean isLeader = true;

for (int i = processId + 1; i < processes.size(); i++) {

if (processes.get(i) > processId) {

System.out.println("Process " + processId + " sends an election message to process " + processes.get(i));

isLeader = false;

}

}

if (!isLeader) {

for (int i = processId + 1; i < processes.size(); i++) {

if (processes.get(i) > processId) {

System.out.println("Process " + processes.get(i) + " acknowledges the election message from process " + processId);

BullyAlgorithm subBully = new BullyAlgorithm(processes.get(i), processes);

int subLeader = subBully.run();

if (subLeader != processes.get(i)) {

System.out.println("Process " + processes.get(i) + " forwards the leader message to process " + processId);

return subLeader;

}

}

}

}

System.out.println("Process " + processId + " is the leader");

return processId;

}

public static void main(String[] args) {

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

processes.add(3);

processes.add(7);

processes.add(2);

processes.add(5);

processes.add(1);

processes.add(9);

BullyAlgorithm bully = new BullyAlgorithm(2, processes);

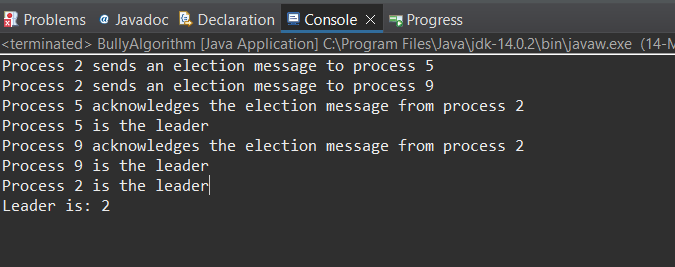
int leader = bully.run();

System.out.println("Leader is: " + leader);

}

}

**OUTPUT:**



**CODE: (Ring Algorithm)**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Scanner;

public class RingAlgorithm {

public static void main(String[] args) {

int temp, i, j;

List<Process> processes = new ArrayList<Process>();

// scanner used for getting input from console

Scanner in = new Scanner(System.in);

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

int num = in.nextInt();

// getting input from users

for (i = 0; i < num; i++) {

System.out.println("Enter the id of process " + (i + 1) + ": ");

int id = in.nextInt();

processes.add(new Process(i, id, "active"));

}

// sorting the processes based on id

Collections.sort(processes);

System.out.println("The sorted processes are:");

for (i = 0; i < num; i++) {

System.out.println("[" + processes.get(i).index + "] " + processes.get(i).id);

}

processes.get(num - 1).state = "inactive";

System.out.println("Process " + processes.get(num - 1).id + " selected as coordinator");

while (true) {

System.out.println("\n1. Election\n2. Quit");

int ch = in.nextInt();

switch (ch) {

case 1:

System.out.println("Enter the process number who initiates the election: ");

int init = in.nextInt();

int temp2 = init;

int temp1 = (init + 1) % num;

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

i = 0;

while (temp2 != temp1) {

Process p = processes.get(temp1);

if ("active".equals(p.state) && p.f == 0) {

System.out.println("Process " + processes.get(init).id + " sends message to process " + p.id);

p.f = 1;

init = temp1;

arr.add(p.id);

i++;

}

temp1 = (temp1 + 1) % num;

}

Process p = processes.get(temp1);

System.out.println("Process " + processes.get(init).id + " sends message to process " + p.id);

arr.add(p.id);

i++;

int max = Collections.max(arr);

System.out.println("Process " + max + " selected as coordinator");

for (Process process : processes) {

if (process.id == max) {

process.state = "inactive";

}

}

break;

case 2:

System.out.println("Program terminated ...");

return;

default:

System.out.println("Invalid response");

break;

}

}

}

}

class Process implements Comparable<Process> {

public int index; // to store the index of process

public int id; // to store id/name of process

public int f;

String state; // indicates whether active or inactive state of node

public Process(int index, int id, String state) {

this.index = index;

this.id = id;

this.state = state;

}

@Override

public int compareTo(Process o) {

return Integer.compare(this.id, o.id);

}

}

**OUPTUT:**

