import java.util.Scanner;

import java.util.Random;

class BullyAlgorithm {

static int numProcesses;

static boolean[] alive;

static int coordinator;

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

Scanner sc = new Scanner(System.in);

Random rand = new Random();

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

numProcesses = sc.nextInt();

alive = new boolean[numProcesses];

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

alive[i] = true; // Initially, all processes are alive

}

// Initial coordinator is the highest-numbered process

coordinator = numProcesses - 1;

System.out.println("Initial Coordinator: Process " + coordinator);

// Randomly crash a process (excluding the highest process)

int crash = rand.nextInt(numProcesses - 1);

alive[crash] = false;

System.out.println("Process " + crash + " has crashed.");

System.out.print("Enter the process to start the election: ");

int initiator = sc.nextInt();

elect(initiator);

sc.close();

}

static void elect(int initiator) throws InterruptedException {

if (!alive[initiator]) {

System.out.println("Process " + initiator + " is crashed and cannot start an election.");

return;

}

System.out.println("\nProcess " + initiator + " is initiating an election...");

boolean higherExists = false;

for (int i = initiator + 1; i < numProcesses; i++) {

if (alive[i]) {

System.out.println("Process " + initiator + " -> Process " + i + " (ELECTION)");

higherExists = true;

}

}

if (!higherExists) {

declareWinner(initiator);

} else {

Thread.sleep(500); // Wait before sending OK responses

for (int i = initiator + 1; i < numProcesses; i++) {

if (alive[i]) {

System.out.println("Process " + i + " -> Process " + initiator + " (OK)");

}

}

for (int i = initiator + 1; i < numProcesses; i++) {

if (alive[i]) {

elect(i);

return;

}

}

}

}

static void declareWinner(int winner) throws InterruptedException {

coordinator = winner;

System.out.println("\nProcess " + winner + " wins the election and becomes the new coordinator.");

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

if (i != winner && alive[i]) {

System.out.println("Process " + winner + " -> Process " + i + " (COORDINATOR)");

Thread.sleep(500);

}

}

}

}

PS C:\Users\Arati\Downloads\BE-IT-DS-main\BE-IT-DS-main\Assign6> java BullyAlgorithm

Enter the number of processes: 8

Initial Coordinator: Process 7

Process 5 has crashed.

Enter the process to start the election: 2

Process 2 is initiating an election...

Process 2 -> Process 3 (ELECTION)

Process 2 -> Process 4 (ELECTION)

Process 2 -> Process 6 (ELECTION)

Process 2 -> Process 7 (ELECTION)

Process 3 -> Process 2 (OK)

Process 4 -> Process 2 (OK)

Process 6 -> Process 2 (OK)

Process 7 -> Process 2 (OK)

Process 3 is initiating an election...

Process 3 -> Process 4 (ELECTION)

Process 3 -> Process 6 (ELECTION)

Process 3 -> Process 7 (ELECTION)

Process 4 -> Process 3 (OK)

Process 6 -> Process 3 (OK)

Process 7 -> Process 3 (OK)

Process 4 is initiating an election...

Process 4 -> Process 6 (ELECTION)

Process 4 -> Process 7 (ELECTION)

Process 6 -> Process 4 (OK)

Process 7 -> Process 4 (OK)

Process 6 is initiating an election...

Process 6 -> Process 7 (ELECTION)

Process 7 -> Process 6 (OK)

Process 7 is initiating an election...

Process 7 wins the election and becomes the new coordinator.

Process 7 -> Process 0 (COORDINATOR)

Process 7 -> Process 1 (COORDINATOR)

Process 7 -> Process 2 (COORDINATOR)

Process 7 -> Process 3 (COORDINATOR)

Process 7 -> Process 4 (COORDINATOR)

Process 7 -> Process 6 (COORDINATOR)

PS C:\Users\Arati\Downloads\BE-IT-DS-main\BE-IT-DS-main\Assign6>

output:

RING ALGORITHM

import java.util.Scanner;

import java.util.ArrayList;

import java.util.List;

class RingElection {

private int numProcesses;

private int coordinator;

private boolean[] activeProcesses;

public RingElection(int numProcesses) {

this.numProcesses = numProcesses;

this.activeProcesses = new boolean[numProcesses];

// Initially, all processes are active

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

activeProcesses[i] = true;

}

// Assume the highest-numbered process is the initial coordinator

coordinator = numProcesses - 1;

System.out.println("Initial Coordinator: Process " + coordinator);

}

// Simulate process crash

public void simulateCrash(int processId) {

if (processId >= 0 && processId < numProcesses && processId != coordinator) {

activeProcesses[processId] = false; // Process is considered crashed

System.out.println("Process " + processId + " has crashed!");

} else {

System.out.println("Invalid choice. This process cannot be crashed.");

}

}

public void startElection(int initiator) {

if (!activeProcesses[initiator]) {

System.out.println("Process " + initiator + " is crashed and cannot start an election.");

return;

}

System.out.println("\nProcess " + initiator + " is initiating an election...");

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

electionPath.add(initiator);

System.out.println("Election path: " + electionPath);

int maxId = initiator;

int current = (initiator + 1) % numProcesses;

while (current != initiator) {

if (activeProcesses[current]) {

System.out.println("Process " + maxId + " -> Process " + current + " (ELECTION)");

electionPath.add(current);

System.out.println("Election path: " + electionPath);

if (current > maxId) {

maxId = current;

}

} else {

System.out.println("Process " + current + " is skipped (CRASHED).");

}

current = (current + 1) % numProcesses;

}

// The highest ID process becomes the coordinator

coordinator = maxId;

System.out.println("\nProcess " + coordinator + " wins the election and becomes the new coordinator.");

announceNewCoordinator();

}

private void announceNewCoordinator() {

int current = (coordinator + 1) % numProcesses;

while (current != coordinator) {

if (activeProcesses[current]) {

System.out.println("Process " + coordinator + " -> Process " + current + " (ELECTED)");

}

current = (current + 1) % numProcesses;

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read number of processes from the user

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

int numProcesses = scanner.nextInt();

// Initialize the RingElection with the number of processes

RingElection ringElection = new RingElection(numProcesses);

// Ask the user which process to crash

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

int crashProcess = scanner.nextInt();

while (crashProcess < 0 || crashProcess >= numProcesses || crashProcess == (numProcesses - 1)) {

System.out.print("Invalid choice. Enter a valid process to crash: ");

crashProcess = scanner.nextInt();

}

ringElection.simulateCrash(crashProcess);

// Read the initiator process that starts the election

System.out.print("\nEnter the process to start the election: ");

int initiator = scanner.nextInt();

while (initiator < 0 || initiator >= numProcesses || !ringElection.activeProcesses[initiator]) {

System.out.print("Invalid or crashed process. Enter a valid process to start the election: ");

initiator = scanner.nextInt();

}

ringElection.startElection(initiator);

}

}

OUTPUT:

PS C:\Users\Arati\Downloads\BE-IT-DS-main\BE-IT-DS-main\Assign6> javac RingElection.java

PS C:\Users\Arati\Downloads\BE-IT-DS-main\BE-IT-DS-main\Assign6> java RingElection

Enter the number of processes: 7

Initial Coordinator: Process 6

Enter a process to crash: 3

Process 3 has crashed!

Enter the process to start the election: 2

Process 2 is initiating an election...

Election path: [2]

Process 3 is skipped (CRASHED).

Process 2 -> Process 4 (ELECTION)

Election path: [2, 4]

Process 4 -> Process 5 (ELECTION)

Election path: [2, 4, 5]

Process 5 -> Process 6 (ELECTION)

Election path: [2, 4, 5, 6]

Process 6 -> Process 0 (ELECTION)

Election path: [2, 4, 5, 6, 0]

Process 6 -> Process 1 (ELECTION)

Election path: [2, 4, 5, 6, 0, 1]

Process 6 wins the election and becomes the new coordinator.

Process 6 -> Process 0 (ELECTED)

Process 6 -> Process 1 (ELECTED)

Process 6 -> Process 2 (ELECTED)

Process 6 -> Process 4 (ELECTED)

Process 6 -> Process 5 (ELECTED)