**Assignment 06:**

**Implement Bully and Ring algorithm for leader election.**

**Bully.java**

**import** java.util.Scanner;

**public** **class** Bully { **static** **boolean**[] *state* = **new** **boolean**[5];

**public** **static** **int** *coordinator* = 4;

**public** **static** **void** getStatus() {

System.***out***.println("\n+------Current System State-----+");

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

System.***out***.println("| P" + (i + 1) + ":\t" + (*state*[i] ? "UP" : "DOWN") + (*coordinator* == i ? "\t<-- COORDINATOR\t|" : "\t\t\t|"));

}

System.***out***.println("+-------------------------------+"); }

**public** **static** **void** up(**int** up) {

**if** (*state*[up - 1]) {

System.***out***.println("Process " + up + " is already up");

} **else** {

*state*[up - 1] = **true**;

System.***out***.println("--------Process " + up + " held election-------"); **for** (**int** i = up; i < *state*.length; ++i) {

System.***out***.println("Election message sent from process " + up + " to process " + (i + 1));

}

**for** (**int** i = *state*.length - 1; i >= 0; --i) { **if** (*state*[i]) { *coordinator* = i; **break**;

}

}

}

}

**public** **static** **void** down(**int** down) {

**if** (!*state*[down - 1]) {

System.***out***.println("Process " + down + " is already down.");

} **else** {

*state*[down - 1] = **false**; **if** (*coordinator* == down - 1) { *setCoordinator*();

}

}

}

**public** **static** **void** mess(**int** mess) {

**if** (*state*[mess - 1]) { **if** (*state*[*coordinator*]) {

System.***out***.println("Message Sent: Coordinator is alive");

} **else** {

System.***out***.println("Coordinator is down");

System.***out***.println("Process " + mess + " initiated election");

**for** (**int** i = mess; i < *state*.length; ++i) {

System.***out***.println("Election sent from process " + mess + " to process " + (i + 1));

}

*setCoordinator*();

}

} **else** {

System.***out***.println("Process " + mess + " is down");

}

}

**public** **static** **void** setCoordinator() { **for** (**int** i = *state*.length - 1; i >= 0; i--) { **if** (*state*[i]) { *coordinator* = i; **break**;

}

}

}

**public** **static** **void** main(String[] args) { **int** choice;

Scanner sc = **new** Scanner(System.***in***); **for** (**int** i = 0; i < *state*.length; ++i) {

*state*[i] = **true**;

} *getStatus*(); **do** {

System.***out***.println("+........MENU........+");

System.***out***.println("1. Activate a process.");

System.***out***.println("2. Deactivate a process.");

System.***out***.println("3. Send a message.");

System.***out***.println("4. Exit.");

System.***out***.println("+....................+"); choice = sc.nextInt(); **switch** (choice) {

**case** 1: {

System.***out***.println("Activate process:");

**int** up = sc.nextInt(); **if** (up == 5) {

System.***out***.println("Process 5 is the coordinator"); *state*[4] = **true**; *coordinator* = 4; **break**; } *up*(up); **break**; }

**case** 2: {

System.***out***.println("Deactivate process:");

**int** down = sc.nextInt(); *down*(down); **break**; } **case** 3: {

System.***out***.println("Send message from process:");

**int** mess = sc.nextInt(); *mess*(mess); **break**;

} } *getStatus*(); } **while** (choice != 4);

sc.close();

}

}

**Output:**

C:\Users\aarad\eclipse-workspace\Assignment6\src>javac Bully.java

C:\Users\aarad\eclipse-workspace\Assignment6\src>java Bully

+------Current System State-----+

| P1: UP |

| P2: UP |

| P3: UP |

| P4: UP |

| P5: UP <-- COORDINATOR |

+-------------------------------+

+........MENU........+

1. Activate a process.
2. Deactivate a process.
3. Send a message.
4. Exit.

+....................+

2

Deactivate process:

5

+------Current System State-----+

| P1: UP |

| P2: UP |

| P3: UP |

| P4: UP <-- COORDINATOR |

| P5: DOWN |

+-------------------------------+

+........MENU........+

1. Activate a process.
2. Deactivate a process.
3. Send a message.
4. Exit.

+....................+

+........MENU........+

+........MENU........+

1. Activate a process.
2. Deactivate a process.
3. Send a message.
4. Exit.

+....................+

3

Send message from process:

2

Message Sent: Coordinator is alive

+------Current System State-----+

| P1: UP |

| P2: UP |

| P3: UP |

| P4: UP <-- COORDINATOR |

| P5: DOWN |

+-------------------------------+

+........MENU........+

1. Activate a process.
2. Deactivate a process.
3. Send a message.
4. Exit.

+....................+

Ring.java import java.util.Scanner; public class Ring { public static void main(String[] args) { Scanner in = new Scanner(System.in);

System.out.println("Enter the number of processes: "); int num = in.nextInt();

Rr[] proc = new Rr[num];

// Initialize processes

// This code block is initializing the processes. It creates an array of Rr objects with a size of

// `num` (which is the number of processes entered by the user), and then prompts the user to enter

// the ID of each process. It sets the index of each process to its corresponding index in the // array, sets the state of each process to "active", and sets the value of `f` (which is used as a

// flag during the election process) to 0 for each process. for (int i = 0; i < num; i++) { proc[i] = new Rr(); proc[i].index = i;

System.out.println("Enter the ID of process " + (i + 1) + ": "); proc[i].id = in.nextInt(); proc[i].state = "active";

proc[i].f = 0;

}

// Sort processes based on ID

// This code block is sorting the `proc` array of `Rr` objects based on the `id` field of each // object. It uses a bubble sort algorithm, where it compares adjacent elements in the array and

// swaps them if they are in the wrong order. The outer loop iterates `num - 1` times, and the

// inner loop iterates `num - 1` times as well. The `if` statement inside the inner loop checks

// if the `id` of the current element is greater than the `id` of the next element. If it is, // then it swaps the two elements using a temporary variable `temp`. This process continues until

// the array is sorted in ascending order based on the `id` field. for (int i = 0; i < num - 1; i++) { for (int j = 0; j < num - 1; j++) { if (proc[j].id > proc[j + 1].id) { Rr temp = proc[j]; proc[j] = proc[j + 1]; proc[j + 1] = temp;

}

}

}

// Print the sorted processes

// This code block is printing out the sorted processes in the `proc` array of `Rr` objects.

It

// uses a `for` loop to iterate through each element in the array, and prints out the index of

// the element (`i`), the `id` field of the `Rr` object at that index (`proc[i].id`), and a // space character. The output is formatted as `[index] id `, where `index` is the index of the

// process in the array, and `id` is the ID/name of the process. for (int i = 0; i < num; i++) {

System.out.print("[" + i + "] " + proc[i].id + " ");

}

// Select last process as coordinator proc[num - 1].state = "inactive";

System.out.println("\nProcess " + proc[num - 1].id + " selected as coordinator");

// This code block is implementing a loop that repeatedly prompts the user to choose between two

// options: initiating an election or quitting the program. It uses a `while` loop with a // condition of `true`, which means that the loop will continue indefinitely until it is // explicitly broken out of using a `return` statement.

while (true) {

System.out.println("\n1. Election\n2. Quit"); int ch = in.nextInt();

// Reset flags

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

proc[i].f = 0;

}

switch (ch) { case 1:

System.out.println("Enter the process number that initializes the election: "); int init = in.nextInt(); int temp2 = init; int temp1 = init + 1;

int i = 0;

while (temp2 != temp1) { if (temp1 == num) { temp1 = 0;

}

if ("active".equals(proc[temp1].state) && proc[temp1].f == 0) {

System.out.println("Process " + proc[init].id + " sends a message to Process

" + proc[temp1].id);

proc[temp1].f = 1; init = temp1;

i++; }

temp1++;

}

System.out.println("Process " + proc[init].id + " sends a message to Process " + proc[temp1].id);

int max = -1;

// Find maximum ID for coordinator selection

for (int j = 0; j < i; j++) { if (max < proc[j].id) { max = proc[j].id;

}

}

// Select coordinator and update states

System.out.println("Process " + max + " selected as coordinator"); for (int k = 0; k < num; k++) { if (proc[k].id == max) {

proc[k].state = "inactive";

} } break; case 2:

System.out.println("Program terminated."); in.close(); return; default:

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

}

}

}

}

class Rr { public int index; // To store the index of the process public int id; // To store the ID/name of the process public int f;

public String state; // Indicates whether the process is active or inactive

}

**Output:**

C:\Users\aarad\eclipse-workspace\Assignment6\src>javac Ring.java

C:\Users\aarad\eclipse-workspace\Assignment6\src>java Ring Enter the number of processes:

5

Enter the ID of process 1:

101

Enter the ID of process 2:

102

Enter the ID of process 3:

103

Enter the ID of process 4:

104

Enter the ID of process 5:

105

[0] 101 [1] 102 [2] 103 [3] 104 [4] 105

Process 105 selected as coordinator

1. Election
2. Quit

1

Enter the process number that initializes the election:

102

4

Process 105 sends a message to Process 101

Process 101 sends a message to Process 102

Process 102 sends a message to Process 103

Process 103 sends a message to Process 104

Process 104 sends a message to Process 105

Process 104 selected as coordinator

1. Election
2. Quit