**Name : Rohit Mahadev Mane Roll No : CO313**

**Class : TE COMP**

Assignment No:5

**Aim:** Write a java program (using OOP features) to implement following scheduling algorithms:

FCFS,SJF(Preemptive),Priority(Non- Preemptive) and Round Robin(Preemptive).

# FCFS:

import java.text.ParseException; class FCFS {

static void findWaitingTime(int processes[], int n, int bt[], int wt[]) {

// waiting time for first process is 0 wt[0] = 0;

// calculating waiting time for (int i = 1; i < n; i++) {

wt[i] = bt[i - 1] + wt[i - 1];

}

}

static void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {

// bt[i] + wt[i]

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

tat[i] = bt[i] + wt[i];

}

}

static void findavgTime(int processes[], int n, int bt[]) { int wt[] = new int[n], tat[] = new int[n];

int total\_wt = 0, total\_tat = 0;

findWaitingTime(processes, n, bt, wt);

findTurnAroundTime(processes, n, bt, wt, tat);

System.out.printf("Processes Burst time Waiting"

+" time Turn around time\n");

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

total\_wt = total\_wt + wt[i]; total\_tat = total\_tat + tat[i]; System.out.printf(" %d ", (i + 1));

System.out.printf(" %d ", bt[i]);

System.out.printf(" %d", wt[i]);

System.out.printf(" %d\n", tat[i]);

}

float s = (float)total\_wt /(float) n; int t = total\_tat / n;

System.out.printf("Average waiting time = %f", s); System.out.printf("\n");

System.out.printf("Average turn around time = %d ", t);

}

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

int processes[] = {1, 2, 3}; int n = processes.length;

int burst\_time[] = {10, 5, 8}; findavgTime(processes, n, burst\_time);

}

}

**Output:**

Microsoft Windows [Version 10.0.22621.755]

(c) Microsoft Corporation. All rights reserved.

C:\Users\Vishnu>d:

D:\>cd spos\5 D:\SPOS\5>javac FCFS.java

D:\SPOS\5>java FCFS

Processes Burst time Waiting time Turn around time 1 10 0 10

2 5 10 15

3 8 15 23

Average waiting time = 8.333333 Average turn around time = 16 D:\SPOS\5>

# SJF:

import java.util.\*;

public class SJF {

public static void main(String args[])

{

Scanner sc = new Scanner(System.in); System.out.println ("enter no of process:"); int n = sc.nextInt();

int process\_id[] = new int[n];

int complete\_time[] = new int[n]; int arrival\_time[] = new int[n];

int burst\_time[] = new int[n];

int turn\_around\_time[] = new int[n];

int waiting\_time[] = new int[n]; //wt means waiting time

int f[] = new int[n]; // f means it is flag it checks process is completed or not int st=0, total\_processes=0;

float avgwt=0, avgta=0;

for(int i=0;i<n;i++)

{

System.out.println ("enter process " + (i+1) + " arrival time:"); arrival\_time[i] = sc.nextInt();

System.out.println ("enter process " + (i+1) + " brust time:"); burst\_time[i] = sc.nextInt();

process\_id[i] = i+1; f[i] = 0;

}

boolean a = true; while(true)

{

int c=n, min=999;

if (total\_processes == n)

break;

for (int i=0; i<n; i++)

{

/\*

* If i'th process arrival time is less than or equal to system time and its flag=0 and burst<min
* this process will be first

\*/

if ((arrival\_time[i] <= st) && (f[i] == 0) && (burst\_time[i]<min))

{

min=burst\_time[i]; c=i;

}

}

/\* If c==n means c value can not updated because no process arrival time< system time so we increase the system time \*/

if (c==n)

st++;

else

{

}

}

complete\_time[c]=st+burst\_time[c]; st+=burst\_time[c];

turn\_around\_time[c]=complete\_time[c]-arrival\_time[c]; waiting\_time[c]=turn\_around\_time[c]-burst\_time[c]; f[c]=1;

total\_processes++;

System.out.println("\npid arrival brust complete turn waiting"); for(int i=0;i<n;i++)

{

avgwt+= waiting\_time[i]; avgta+= turn\_around\_time[i];

System.out.println(process\_id[i]+"\t"+arrival\_time[i]+"\t"+burst\_time[i]+"\t"+compl ete\_time[i]+"\t"+turn\_around\_time[i]+"\t"+waiting\_time[i]);

}

System.out.println ("\naverage tat is "+ (float)(avgta/n)); System.out.println ("average wt is "+ (float)(avgwt/n)); sc.close();

}

}

**Output:**

D:\SPOS\5>javac SJF.java

D:\SPOS\5>java SJF enter no of process: 3

enter process 1 arrival time: 112

enter process 1 brust time: 23

enter process 2 arrival time: 45

enter process 2 brust time:

34

enter process 3 arrival time: 90

enter process 3 brust time:

3

pid arrival brust complete turn waiting

1 112

2 45

3 90

23

34

3

135 23 0

79

93

34 0

3

0

average tat is 20.0

average wt is 0.0

D:\SPOS\5>

# PRIORITY:

import java.util.\*; class Process

{

int pid; // Process ID

int bt; // CPU Burst time required

int priority; // Priority of this process Process(int pid, int bt, int priority)

{

this.pid = pid; this.bt = bt;

this.priority = priority;

}

public int prior() {

return priority;

}

}

public class PRIORITY

{

public void findWaitingTime(Process proc[], int n,

int wt[])

{

// waiting time for first process is 0 wt[0] = 0;

// calculating waiting time for (int i = 1; i < n ; i++ )

wt[i] = proc[i - 1].bt + wt[i - 1] ;

}

// Function to calculate turn around time

public void findTurnAroundTime( Process proc[], int n,

int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

for (int i = 0; i < n ; i++)

tat[i] = proc[i].bt + wt[i];

}

// Function to calculate average time

public void findavgTime(Process proc[], int n)

{

int wt[] = new int[n], tat[] = new int[n], total\_wt = 0, total\_tat = 0;

// Function to find waiting time of all processes findWaitingTime(proc, n, wt);

// Function to find turn around time for all processes findTurnAroundTime(proc, n, wt, tat);

// Display processes along with all details

System.out.print("\nProcesses Burst time Waiting time Turn around time\n");

for (int i = 0; i < n; i++)

{

"\n");

}

total\_wt = total\_wt + wt[i]; total\_tat = total\_tat + tat[i];

System.out.print(" " + proc[i].pid + "\t\t" + proc[i].bt + "\t " + wt[i] + "\t\t " + tat[i] +

System.out.print("\nAverage waiting time = "

+(float)total\_wt / (float)n);

System.out.print("\nAverage turn around time = "+(float)total\_tat / (float)n);

}

public void priorityScheduling(Process proc[], int n)

{

// Sort processes by priority

Arrays.sort(proc, new Comparator<Process>() { @Override

public int compare(Process a, Process b) { return b.prior() - a.prior();

}

});

System.out.print("Order in which processes gets executed \n"); for (int i = 0 ; i < n; i++)

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

findavgTime(proc, n);

}

// Driver code

public static void main(String[] args)

{

PRIORITY ob=new PRIORITY();

int n = 3;

Process proc[] = new Process[n]; proc[0] = new Process(1, 10, 2);

proc[1] = new Process(2, 5, 0);

proc[2] = new Process(3, 8, 1); ob.priorityScheduling(proc, n);

}

}

**Output:**

D:\SPOS\5>javac PRIORITY.java

D:\SPOS\5>java PRIORITY

Order in which processes gets executed 1 3 2

Processes Burst time Waiting time Turn around time

Average waiting time = 9.333333

Average turn around time = 17.0 D:\SPOS\5>

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 10 | 0 | 10 |
| 3 | 8 | 10 | 18 |
| 2 | 5 | 18 | 23 |

# ROUND ROBIN:

public class ROUNDROBIN

{

// Method to find the waiting time for all

// processes

static void findWaitingTime(int processes[], int n,

int bt[], int wt[], int quantum)

{

// Make a copy of burst times bt[] to store remaining

// burst times.

int rem\_bt[] = new int[n]; for (int i = 0 ; i < n ; i++)

rem\_bt[i] = bt[i]; int t = 0; // Current time

// Keep traversing processes in round robin manner

// until all of them are not done. while(true)

{

boolean done = true;

// Traverse all processes one by one repeatedly for (int i = 0 ; i < n; i++)

{

// If burst time of a process is greater than 0

// then only need to process further if (rem\_bt[i] > 0)

{

done = false; // There is a pending process

if (rem\_bt[i] > quantum)

{

t += quantum;

rem\_bt[i] -= quantum;

}

// If burst time is smaller than or equal to

// quantum. Last cycle for this process else

{

t = t + rem\_bt[i];

// used by this process wt[i] = t - bt[i];

// As the process gets fully executed

// make its remaining burst time = 0 rem\_bt[i] = 0;

}

}

}

// If all processes are done if (done == true)

break;

}

}

// Method to calculate turn around time

static void findTurnAroundTime(int processes[], int n,int bt[], int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

for (int i = 0; i < n ; i++)

tat[i] = bt[i] + wt[i];

}

// Method to calculate average time

static void findavgTime(int processes[], int n, int bt[],int quantum)

{

int wt[] = new int[n], tat[] = new int[n]; int total\_wt = 0, total\_tat = 0;

// Function to find waiting time of all processes findWaitingTime(processes, n, bt, wt, quantum);

// Function to find turn around time for all processes findTurnAroundTime(processes, n, bt, wt, tat);

// Display processes along with all details System.out.println("PN " + " B " +" WT " + " TAT");

// Calculate total waiting time and total turn

// around time

for (int i=0; i<n; i++)

{

total\_wt = total\_wt + wt[i]; total\_tat = total\_tat + tat[i];

System.out.println(" " + (i+1) + "\t\t" + bt[i] +"\t " +

wt[i] +"\t\t " + tat[i]);

}

System.out.println("Average waiting time = " +(float)total\_wt / (float)n); System.out.println("Average turn around time = " +(float)total\_tat / (float)n);

}

// Driver Method

public static void main(String[] args)

{

// process id's

int processes[] = { 1, 2, 3}; int n = processes.length;

// Burst time of all processes int burst\_time[] = {10, 5, 8};

// Time quantum int quantum = 2;

findavgTime(processes, n, burst\_time, quantum);

}

}

# Output:

D:\SPOS\5>javac ROUNDROBIN.java

D:\SPOS\5>java ROUNDROBIN

Average waiting time = 12.0

Average turn around time = 19.666666

D:\SPOS\5>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PN | B | WT TAT |  | |
| 1 |  | 10 | 13 | 23 |
| 2 |  | 5 | 10 | 15 |
| 3 |  | 8 | 13 | 21 |