Contents

[USCSP301\_USCS303\_OS\_B1\_RS\_3 2](#_Toc78383539)

[PRACTICAL 3: ROUND-ROBIN SCHEDULING ALGORITHM 2](#_Toc78383540)

[Practical date: 27th July,2021. 2](#_Toc78383541)

[Practical Aim: Implement RR scheduling algorithm in Java. 2](#_Toc78383542)

[ALGORITHM: 2](#_Toc78383543)

[FLOWCHART 3](#_Toc78383544)

[SOLVED EXAMPLES : 4](#_Toc78383545)

[GANTT CHART: 7](#_Toc78383546)

[IMPLEMENTATION: 8](#_Toc78383547)

[EXAMPLE 2 : 12](#_Toc78383548)

[EXAMPLE 3: 13](#_Toc78383549)

[SAMPLE OUTPUT 1: 15](#_Toc78383550)

[SAMPLE OUTPUT 2: 16](#_Toc78383551)

[SAMPLE OUTPUT 3: 16](#_Toc78383552)

# USCSP301\_USCS303\_OS\_B1\_RS\_3

## PRACTICAL 3: ROUND-ROBIN SCHEDULING ALGORITHM

### Practical date: 27th July,2021.

### Practical Aim: Implement RR scheduling algorithm in Java.

### ALGORITHM:

**Round-robin (RR) scheduling algorithm** is mainly designedfor time-sharing systems.

This algorithm is similar to FCFS scheduling,but in Round-robin(RR) scheduling preemption is added which enables the system to switch between processes.

Round-Robin scheduling algorithm is used to schedule process fairly each job a time slot or quantum and the interrupting the job if it is not completed by then the job come after the other job which is arrivrd in the quantum time that makes these scheduling fairly.

### FLOWCHART

RR Flowchart

Termination

Execute till completion

Burst Time

<Time Quantum

Select a process

Yes

No

No

Execute for the Quantum

Process

executed

completely?

Yes

### SOLVED EXAMPLES :

Consider the following example containing three processes arriving at time t=0 m/s.

**Step 1:** Consider the time quanta / time slice =4 ms.

**Step 2:** Following shows the scheduling and execution of processes.

**Step 2.1:**P0 process arrives at 0 with 24 ms as the burst time which is greater than time quanta = 4 ms. So P0 executes for 4 ms ang goes in waiting queue.

|  |  |
| --- | --- |
| System Time | 0 |
| Process Scheduled | P0 |
| Remaining Time | 24-4=20 |
| Waiting Time | 0-0=0 |
| Turn Around Time | 0+4=4 |

**Step 2.2:** Next P1 process executes for 3 ms which is greater than quanta time. So P1 executes and gets terminated.

|  |  |
| --- | --- |
| System Time | 4 |
| Process Scheduled | P0,P1 |
| Remaining Time | 3-4=-1=0 |
| Waiting Time | 4-0=4 |
| Turn Around Time | 4+3=7 |

**Step 2.3:** Next P2 process executes for 3 ms which is greater than quanta time. So P2 executes and gets terminated.

|  |  |
| --- | --- |
| System Time | 7 |
| Process Scheduled | P0,P1,P2 |
| Remaining Time | 3-4=-1=0 |
| Waiting Time | 7-0=7 |
| Turn Around Time | 4+3=7 |

**Step 2.4:** Now P0 turns comes again and it’s the only process for execution for 4 ms of quanta it gets executed.

|  |  |
| --- | --- |
| System Time | 10 |
| Process Scheduled | P0,P1,P2,P0 |
| Remaining Time | 20-4=16 |
| Waiting Time | 0 |
| Turn Around Time | 10+4=14 |

**Step 2.5:** Again,P0 continues to execute for next 4 ms. Waiting for P0 will be zero.

|  |  |
| --- | --- |
| System Time | 14 |
| Process Scheduled | P0,P1,P2,P0,P0 |
| Remaining Time | 16-4=12 |
| Waiting Time | 0 |
| Turn Around Time | 14+4=18 |

**Step 2.6:** P0 continues to execute for next 4 ms.

|  |  |
| --- | --- |
| System Time | 18 |
| Process Scheduled | P0,P1,P2,P0,P0,P0 |
| Finish Time | 12-4=8 |
| Turn Around Time | 18+4=22 |

**Step 2.7:** P0 continues to execute for next 4 ms.

|  |  |
| --- | --- |
| System Time | 22 |
| Process Scheduled | P0,P1,P2,P0,P0,P0,P0 |
| Finish Time | 8-4=4 |
| Turn Around Time | 26+4=30 |

**Step 2.8:** P0 continues to execute for next 4 ms.

|  |  |
| --- | --- |
| System Time | 26 |
| Process Scheduled | P0,P1,P2,P0,P0,P0,P0,P0 |
| Finish Time | 4-4=0 |
| Turn Around Time | 26+4=30 |

**Step 3:** Calculate Average Waiting Time and Average Turn Around Time.

|  |
| --- |
| Average Waiting Time=(6+4+7)/3  =17/3  =5.6667  Average Turn Around Time=(30+7+10)/3  =47/3  =16 |

**Step 4:** After scheduling of all provided processes.

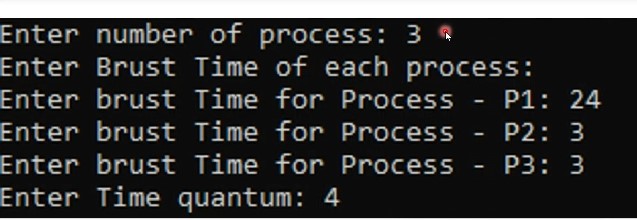
|  |  |  |  |
| --- | --- | --- | --- |
| **Process ID** | **Burst Time** | **Turn Around Time**  **(Completion Time-Arrival Time)** | **Waiting Time**  **(Turn Around Time-Burst Time)** |
| P0 | 24 | 30-0=30 | 30-24=6 |
| P1 | 3 | 4+3=7 | 7-3=4 |
| P2 | 3 | 7+3=10 | 10-3=7 |
| **Average** |  | 15.66667 | 5.6667 |

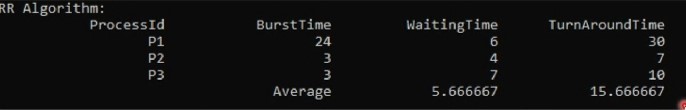
### GANTT CHART:

|  |  |  |  |
| --- | --- | --- | --- |
| **Process ID** | **Burst Time** | **Turn Around Time**  **(Completion Time-Arrival Time)** | **Waiting Time**  **(Turn Around Time-Burst Time)** |
| P0 | 24 | 30-0=30 | 30-24=6 |
| P1 | 3 | 4+3=7 | 7-3=4 |
| P2 | 3 | 7+3=10 | 10-3=7 |
| **Average** |  | 15.6667 | 5.6667 |

|  |
| --- |
| P0 P1 P2 P0 P0 P0 P0 P0 |

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

**INPUT:** ****

**OUTPUT:** ****

**IMPLEMENTATION:**

//Name:Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 27TH JULY, 2021.

//Practical 3: Round-Robin Scheduling Algorithm

import java.util.Scannner;

class P3\_RR\_RS {

public static void main(String args[]) {

Scanner input=new Scanner(System.in);

int i,j,k,q,sum=0;

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

int n= input.nextInt();

int burstTime[]=new int[n];

int waitingTime[]=new int [n];

int turnAroundTime[]=new int [n];

int a[]=new int [n];

System.out.printIn("Enter Burst Time of each process: ");

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

System.out.print("Enter burst Time for Process-P"+(i+1)+":");

burstTime[i]=input.nextInt();

a[i]=burstTime[i];

}

Sytem.out.print("Enter Time quantum: ");

q=input.nextInt();

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

waiting Time[i]=0;

int timer=0;//Current Time

//keep traversing processes in round robin manner

//until all of them are done.

do {

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

//If burst time of a process is greater than 0 then they only need to process further

if (burstTime[i]>q) {

//Increase the value of t i.e. shows how much time fr a process has been processed

timer+=q;

//Decrease the burst time of current process by quantum

burstTime[i]=-q;

for(j=0;j<n;j++) {

if((j!=1)&&(burstTime[j]!=0))

waitingTime[j]=+=q;

}

}// if ends

//If burst time is smaller than or equal to quantum.Last cycle for this process

else {

//Increase the value of t i.e. shows how much time a process has been processed

timer+=burstTime[i];

for(j=0;j<n;j++) {

if((j!=1)&&(burstTime[j]!=0))

waitingTime[j]+=burstTime[i];

}

//As the process get fully executed make its remaining burst time=0

burstTime[i]=0;

}

//else ends

sum=0;for(k=0;k<n;k++)

sum+=burstTime[k];

}while(sum!=0)

//calculating turnaround time by adding waitingTime+burstTime

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

turnAroundTime[i]=waitingTime[i]+a[i];

float total=0;

for(int n:waiting Time) {

total +=n;

}

float averageWaitingTime=total/n;

total=0;

for(int n:turnAroundTime) {

total+=n;

}

float averageTurnAroundTime=total/n;

//print on console the order of processes scheduled using Round-robin Algorithm

System.out.printIn("RR Algorithm");

System.out.format("%20s%20s%20s%20s\n", "ProcessId", "BurstTime", "WaitingTime", "TurnAroundTime");

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

System.out.format("%20s%20d%20d%20d\n", "P"+(i+1), a[i], "WaitingTime", "TurnAroundTime");

}

System.out.format("%40s%20f%20f\n", "Average", averageWaitingTime, averageTurnAroundTime);

}

}

### EXAMPLE 2 :

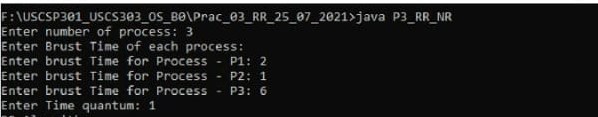
Consider the following example containing three processes arrive at same time having time slice as 1 ms.

|  |  |
| --- | --- |
| **Process ID** | **Burst Time** |
| P0 | 2 |
| P1 | 1 |
| P2 | 6 |

**GANTT CHART:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Process ID** | **Burst Time** | **Turn Around Time** | **Waiting Time** |
| P0 | 2 | 4 | 2 |
| P1 | 1 | 2 | 1 |
| P2 | 6 | 9 | 3 |
| **Average** |  | 5 | 2 |

**INPUT:**

****

**OUTPUT:**

****

**EXAMPLE 3:**

Consider the following example containing three processes arrive at same time. Time Quanta =3.

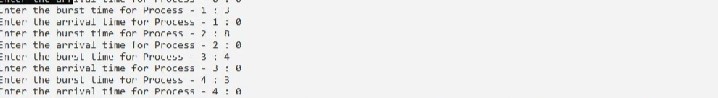
|  |  |
| --- | --- |
| **Process ID** | **Burst Time** |
| P0 | 7 |
| P1 | 3 |
| P2 | 2 |
| P3 | 10 |
| P4 | 8 |

**GANTT CHART:**

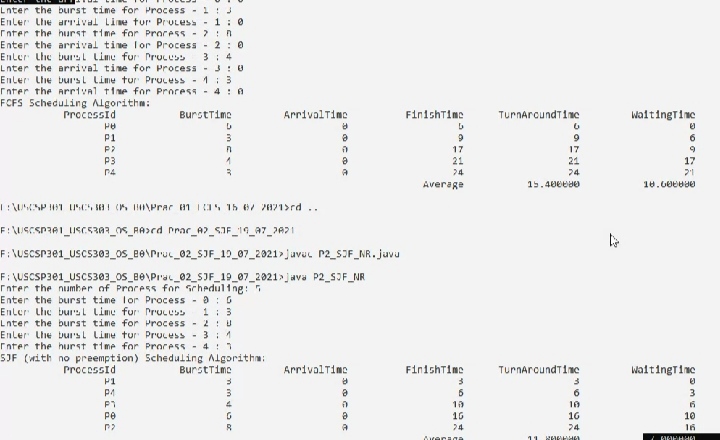
|  |  |  |  |
| --- | --- | --- | --- |
| **Process ID** | **Burst Time** | **Turn Around time** | **Waiting Time** |
| P0 | 7 | 7 | 14 |
| P1 | 3 | 7 | 10 |
| P2 | 2 | 6 | 8 |
| P3 | 10 | 14 | 24 |
| P4 | 8 | 19 | 27 |

**Average:** 11.2 16.6

**INPUT:**

****

**OUTPUT:**

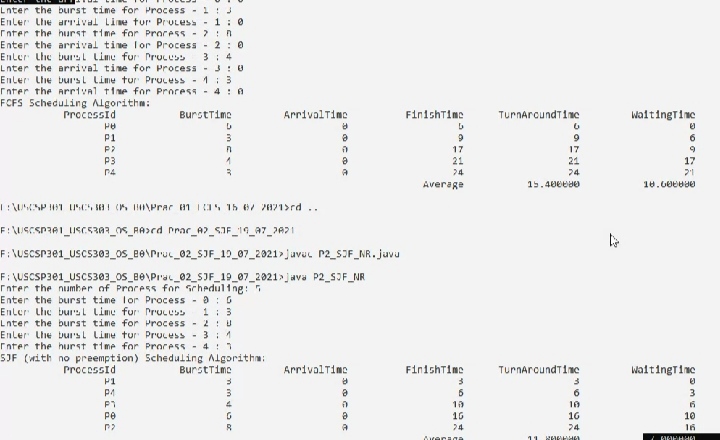
****

### SAMPLE OUTPUT 1: C:\Users\Mithilesh\Desktop\R\SAMPLE 1 OUTPUT.jpeg

### SAMPLE OUTPUT 2:

****

### SAMPLE OUTPUT 3:

****