

Ex. No.: 6d)

Date

ROUND ROBIN SCHEDULING

Aim:

To implement the Round Robin (RR) scheduling technique

Algorithm:

1. Declare the structure and its elements.
2. Get number of processes and Time quantum as input from the user.
3. Read the process name, arrival time and burst time
4. Create an array **rem_bt[]** to keep track of remaining burst time of processes which is initially copy of bt[] (burst times array)
5. Create another array **wt[]** to store waiting times of processes. Initialize this array as 0.
6. Initialize time : $t = 0$
7. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.
 - a- If $\text{rem_bt}[i] > \text{quantum}$
 - (i) $t = t + \text{quantum}$
 - (ii) $\text{bt_rem}[i] -= \text{quantum}$;
 - b- Else // Last cycle for this process
 - (i) $t = t + \text{bt_rem}[i]$;
 - (ii) $\text{wt}[i] = t - \text{bt}[i]$
 - (iii) $\text{bt_rem}[i] = 0$; // This process is over
8. Calculate the waiting time and turnaround time for each process.
9. Calculate the average waiting time and average turnaround time.
10. Display the results.

Program Code:

```
#include<stdio.h>
```

```
int main()
{
    int i, limit, total = 0, x, counter = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10]; float
    average_wait_time, average_turnaround_time;
    printf("\nEnter Total Number of Processes:t");
    scanf("%d", &limit);
    x = limit;
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Details of Process[%d]\n", i + 1);

        printf("Arrival Time:t");

        scanf("%d", &arrival_time[i]);

        printf("Burst Time:t");
```

```

scanf("%d", &burst_time[i]);

temp[i] = burst_time[i];
}

printf("\nEnter Time Quantum:t");
scanf("%d", &time_quantum);
printf("\nProcess ID\tBurst Time\t Turnaround Time\t Waiting Time\n"); for(total
= 0, i = 0; i < limit; i++)
{
    if(temp[i] <= time_quantum && temp[i] > 0)
    {
        total = total + temp[i];
        temp[i] = 0;
        counter = 1;
    }
    else if(temp[i] > 0)
    {
        temp[i] = temp[i] - time_quantum;
        total = total + time_quantum;
    }
    if(temp[i] == 0 && counter == 1)
    {
        x++;
        printf("\nProcess[%d]\tBurst Time\t Turnaround Time\t Waiting Time\t", i + 1, burst_time[i], total - arrival_time[i], total -
arrival_time[i] - burst_time[i]);
        wait_time = wait_time + total - arrival_time[i] - burst_time[i];
        turnaround_time = turnaround_time + total - arrival_time[i]; counter = 0;
    }
    if(i == limit - 1)
    {
        i = 0;
    }
    else if(arrival_time[i + 1] <= total)
    {
        i++;
    }
    else
    {
        i = 0;
    }
}

average_wait_time = wait_time * 1.0 / limit;
average_turnaround_time = turnaround_time * 1.0 / limit;
printf("\nAverage Waiting Time:t%f", average_wait_time);
printf("\nAvg Turnaround Time:t%f", average_turnaround_time);
return 0;

```

}

Output:

```
C:\WINDOWS\SYSTEM32\cmd.exe
Enter Total Number of Processes:      4

Enter Details of Process[1]
Arrival Time:    0
Burst Time:      4

Enter Details of Process[2]
Arrival Time:    1
Burst Time:      7

Enter Details of Process[3]
Arrival Time:    2
Burst Time:      5

Enter Details of Process[4]
Arrival Time:    3
Burst Time:      6

Enter Time Quantum:    3

Process ID          Burst Time      Turnaround Time      Waiting Time
Process[1]          4              13                   9
Process[3]          5              16                   11
Process[4]          6              18                   12
Process[2]          7              21                   14

Average Waiting Time: 11.500000
Avg Turnaround Time:  17.000000
```