Ex. No.: 6d)

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 rem_bt[i] > quantum

(i) t = t + quantum

(ii) bt_rem[i] -= quantum;

b- Else // Last cycle for this process

(i) t = t + bt_rem[i];

(ii) wt[i] = t - bt[i]

(iii) 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>

struct Process { char name[10]; int burst_time; int remaining_burst_time; int
waiting_time; int turnaround_time; };

void roundRobinScheduling(struct Process p[], int n, int quantum) { int total_wt = 0,
total_tat = 0; int t = 0; // Initialize time

// Initialize remaining burst times for each process
for (int i = 0; i < n; i++) {
    p[i].remaining_burst_time = p[i].burst_time;
}

// Round Robin scheduling
while (1) {
    int done = 1;

    // Traverse all processes
    for (int i = 0; i < n; i++) {
        if (p[i].remaining_burst_time > 0) {
            done = 0;  // If any process is still remaining

            // If burst time > quantum, reduce by quantum
            if (p[i].remaining_burst_time > quantum) {
                t += quantum;
                p[i].remaining_burst_time -= quantum;
            }
            // Else, finish the process
            else {
                t += p[i].remaining_burst_time;
                p[i].waiting_time = t - p[i].burst_time;
                p[i].turnaround_time = t;
                total_wt += p[i].waiting_time;
                total_tat += p[i].turnaround_time;
                p[i].remaining_burst_time = 0; // Process is finished
            }
        }
    }
```

```
    // If all processes are done
    if (done == 1)
        break;
}

// Display results
printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
for (int i = 0; i < n; i++) {
    printf("%s\t\t%d\t\t%d\t\t%d\n", p[i].name, p[i].burst_time, p[i].waiting_time,
p[i].turnaround_time);
}

printf("\nAverage Waiting Time: %.2f", (float)total_wt / n);
printf("\nAverage Turnaround Time: %.2f", (float)total_tat / n);


}

int main() { int n, quantum;

// Input the number of processes and the time quantum
printf("Enter the number of processes: ");
scanf("%d", &n);
printf("Enter the time quantum: ");
scanf("%d", &quantum);

struct Process p[n];

// Input process details
for (int i = 0; i < n; i++) {
    printf("\nEnter process name (e.g., P1, P2, etc.): ");
    scanf("%s", p[i].name);
    printf("Enter burst time for %s: ", p[i].name);
    scanf("%d", &p[i].burst_time);
}

// Perform Round Robin scheduling
roundRobinScheduling(p, n, quantum);

return 0;
```

}

INPUT:

Enter the number of processes: 3

Enter the time quantum: 4

Enter process name (e.g., P1, P2, etc.): P1

Enter burst time for P1: 12

Enter process name (e.g., P1, P2, etc.): P2

Enter burst time for P2: 6

Enter process name (e.g., P1, P2, etc.): P3

Enter burst time for P3: 8

OUTPUT:

Process	Burst Time	Waiting Time	Turnaround Time
P1	12	6	18
P2	6	4	10
P3	8	6	14

Average Waiting Time: 5.33

Average Turnaround Time: 14.00

RESULT:

Round robin technique is implemented successfully.