Experiment No: 10
Experiment Name: Implementation of Round Robin Scheduling Algorithm.
Objectives:

- What is Round Robin Scheduling Algorithm?
- How to implement in c?

❖ Round Robin Scheduling Algorithm

What is round robin scheduling algorithm?

→ Round Robin is the preemptive process scheduling algorithm
→ Each process is provided a fix time to execute, it is called a quantum.
→ Once a process is execute for a given time period, it is preempted and other process executes for a given time period.

Aim: To write a c program to implement the CPU scheduling round robin algorithm.

Description:

To aim is to calculate the average waiting time. There will be a time slice, each process should be executed within that time-slice and if not it will go to the waiting state so first check whether the burst time is less than the time-slice. If it is less than it assign the waiting time to the sum of the total times. If it is greater than the burst-time then subtract the time slot from the actual burst time and increment it by time-slot and the loop continues until all the processes are completed.

Algorithm:
Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Calculate the no. of time slices for each process where
No. of time slice for process(n) = burst time process(n)/time slice
Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

a. Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU from process(n-1)
b. Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

a. Average waiting time = Total waiting Time / Number of process
b. Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

Source Code:

```
#include<stdio.h>
void main()
{
    int n,i,j,t,ct[30],bt[30],max,wt[30],tat[30];
    float awt,atat,temp=0;
    printf("Enter the num of process : ");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("\nEnter the burst time P%d = ",i+1);
        scanf("%d",&bt[i]);
        ct[i]=bt[i];
    }
    printf("\nEnter the time quantum is : ");
    scanf("%d",&t);

    max = bt[0];
    for(i=1; i<n; i++)
    {
        if(max<bt[i])
        {
            max=bt[i];
```

```
    }
}
for(i=0; i<(max/t)+1; i++)
{
    for(j=0; j<n; j++)
    {
        if(bt[j]!=0)
        {
            if(bt[j]<=t)
            {
                tat[j]=temp+bt[j];
                temp=temp+bt[j];
                bt[j]=0;
            }
            else
            {
                bt[j]=bt[j]-t;
                temp=temp+t;
            }
        }
    }
}
printf("\n\tprocess \t burst time \t waiting time \t turnaround time\n");
awt=atat=0;
for(i=0; i<n; i++)
{
    wt[i]=tat[i]-ct[i];
    atat= atat+tat[i];
    awt=awt+wt[i];
}
for(i=0; i<n; i++)
{
    printf("\tP%d \t\t %d  \t\t %d \t\t %d\n",i+1,ct[i],wt[i],tat[i]);
}
awt=awt/n;
atat=atat/n;
printf("\nAverage waiting time %.2f\n",awt);
printf("Average turnaround time %.2f\n",atat);
```

```
    return 0;
}
```

Output:

```
E:\programming\OS_Lab\RR_Algorithm.exe
Enter the num of process : 4

Enter the burst time P1 = 5

Enter the burst time P2 = 3

Enter the burst time P3 = 1

Enter the burst time P4 = 4

Enter the time quantum is : 2

        process          burst time      waiting time    turnaround time
        P1               5               8               13
        P2               3               7               10
        P3               1               4               5
        P4               4               8               12

Average waiting time 6.75
Average turnaround time 10.00

Process returned 30 (0x1E)   execution time : 37.240 s
Press any key to continue.
```

Result:

Thus the Round Robin scheduling program was executed and verified successfully.