Lab Report No. 10

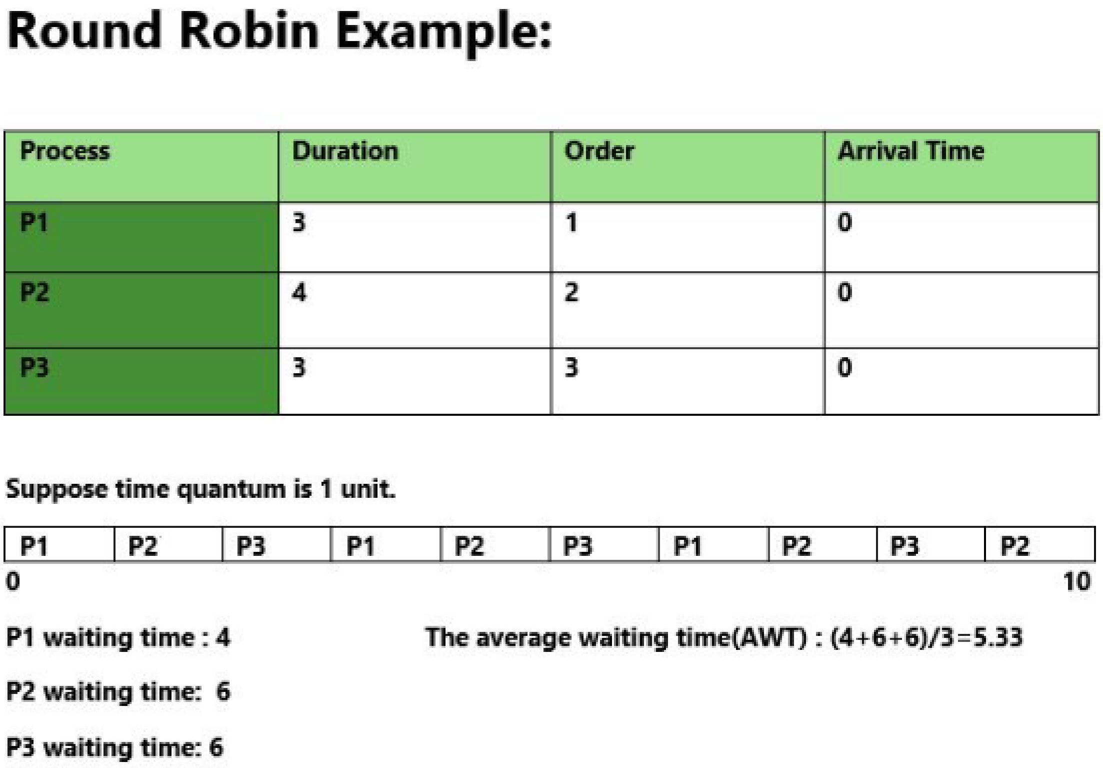
Lab Report Name: Implementation of Round Robin Scheduling algorithm .
Objectives:

i. What is Round Robin Scheduling algorithm.
ii. How to implementation

Theory:

- ➢ 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 executed for a given time period, it is preempted and other process executes for a given time period.
- ➢ Context switching is used to save states of preempted processes.

Example:



Round Robin Example:

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is 1 unit.

P1	P2	P3	P1	P2	P3	P1	P2	P3	P2

0 10

P1 waiting time : 4

P2 waiting time: 6

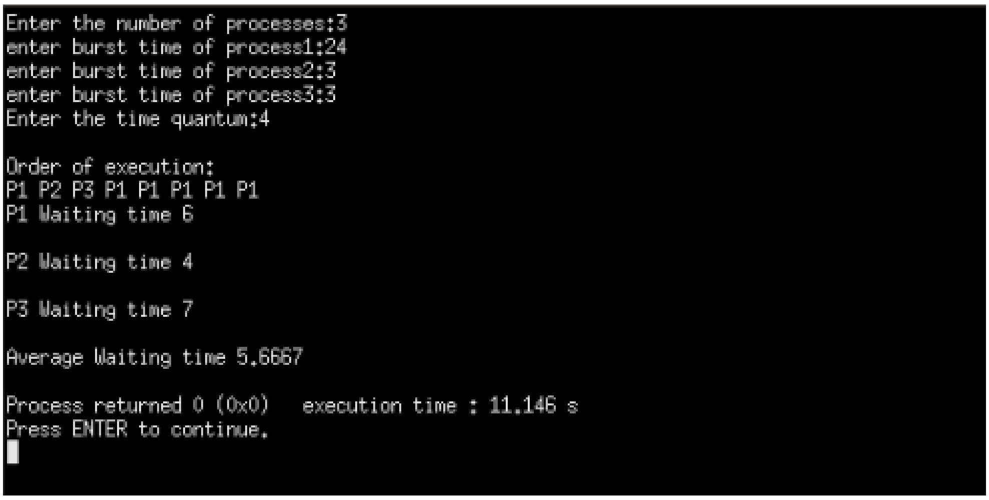
P3 waiting time: 6

The average waiting time(AWT) : (4+6+6)/3=5.33

C program to implement Round Robin algorithms:

```
#include<bits/stdc++.h>
using namespace std;

 main()
{
int n;

    printf("Enter the number of processes:");
    scanf("%d",&n);
    int b[n],a[n],w[n],p[n],tq,flag=0,i,k,temp[100],total=0;
    for(i=0; i<n; i++)
    {
        printf("enter burst time of process%d:",i+1);
        scanf("%d",&b[i]);
        w[i]=0;
        p[i]=0;

    }

    printf("Enter the time quantum:");
    scanf("%d",&tq);
    i=0;
    k=0;

    while(1)
    {
        if(i==n)
            i=0;
        if(flag==n)
            break;


        if(b[i]<=tq && b[i]>0)
        {
            w[i]+=total-p[i];
            flag++;

            total+=b[i];
            b[i]=0;
            temp[k]=i;
            k++;

        }
        if(b[i]>tq)
        {
            w[i]+=total-p[i];
            b[i]=b[i]-tq;
            total+=tq;
            temp[k]=i;
            k++;
        }
        p[i]=total;
        i++;
    }
    printf("\nOrder of execution:\n");
    for(i=0; i<k; i++)
    {
        printf("P%d ",temp[i]+1);
    }


    int a_w_time=0;
    for(i=0; i<n; i++)
    {
        printf("\nP%d Waiting time %d\n",i+1,w[i]);
        a_w_time = a_w_time + w[i];

    }
    printf("\nAverage Waiting time %.4f\n",((float)a_w_time/(float)n));

}
```

outputs:



```
Enter the number of processes:3
enter burst time of process1:24
enter burst time of process2:3
enter burst time of process3:3
Enter the time quantum:4

Order of execution:
P1 P2 P3 P1 P1 P1 P1 P1
P1 Waiting time 6

P2 Waiting time 4

P3 Waiting time 7

Average Waiting time 5.6667

Process returned 0 (0x0)   execution time : 11.146 s
Press ENTER to continue.
```