c) Round Robin

Theory:

Round Robin: It is a primitive scheduling algorithm. it is designed especially for time sharing systems. In this, the CPU switches between the processes. When the time quantum expired, the CPU switches to another job. A small unit of time called a quantum or time slice. A time quantum is generally being a circular queue new processes are added to the tail of the ready queue.

If the process may have a CPU burst of less than one time slice then the process release the CPU voluntarily. The scheduler will then process to next process ready queue otherwise; the process will be put at the tail of the ready queue.

Algorithm

1: Start the process

2: Accept the number of processes in the ready Queue and time quantum (or) time slice

3: For each process in the ready Q, assign the process id and accept the CPU burst time

4: Calculate the no. of time slices for each process where

No. of time slice for process(n) = burst time process(n)/time slice

5: If the burst time is less than the time slice then the no. of time slices =1.

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).

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

Code:

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

```
void main()
{
   int p[20],bt[20]={0},tat[20]={0},wt[20]={0},i,temp=0,ct[20]={0},max,stat=0,swt=0,n,tmslc,j;
   float awt, atat;
printf("Enter the no of processes:");
scanf("%d",&n);
printf("Enter the processes number:");
        for(i=0;i<n;i++)
        {
          scanf("%d",&p[i]);
        }
printf("Enter the  burst times:");
        for(i=0;i<n;i++)
        {
                        scanf("%d",&bt[i]);
                        ct[i]=bt[i];
        }
printf("Enter the tmslc:");
scanf("%d",&tmslc);
        max=bt[0];
        for(i=1;i<n;i++)
        {
            if(max<bt[i])
             max=bt[i];
        }
        for(j=0;j<(max/tmslc)+1;j++)
        {
            for(i=0;i<n;i++)
```

```
            {
                if(bt[i]!=0)
                {
                    if(bt[i]<=tmslc)
                    {
                            tat[i]=temp+bt[i];
                            temp=temp+bt[i];
                            bt[i]=0;
                    }
                    else
                    {
                        bt[i]=bt[i]-tmslc;
                        temp=temp+tmslc;
                    }
                }
            }
        }
        for(i=0;i<n;i++)
        {
          wt[i]=tat[i]-ct[i];
          stat+=tat[i];
          swt+=wt[i];
        }
atat=(float)stat/n;
awt=(float)swt/n;
printf("Process\tBT\tWT\tTAT");
printf("\n---------------------------------------");
        for(i=0;i<n;i++)
```

```
        {
          printf("\n P%d  \t%d\t%d \t%d",p[i],ct[i],wt[i],tat[i]);
        }
printf("\n---------------------------------------");
printf("\n  \t(AWT)%f   %f(ATAT)\n",awt,atat);
printf("\n---------------------------------------");
}
```

OUTPUT:

```
Enter the no of processes:3
Enter the processes number:1
2
3
Enter the burst times:24
3
3
Enter the tmslc:4
Process BT      WT      TAT
---------------------------------------
 P1     24      6       30
 P2     3       4       7
 P3     3       7       10
---------------------------------------
 (AWT)5.666667   15.666667(ATAT)
---------------------------------------
--------------------------------
Process exited after 14.21 seconds with return value 40
Press any key to continue . . .
```

(d) Priority:

Theory:

Priority Scheduling: A Priority is associated with each process, and the cpu is allocated to the process with the highest priority. The cpu is allocated to the process with the highest priority. Equal priority processes are scheduled in the FCFS order. Priorities are generally some fixed range of numbers such as 0 to 409. The low numbers represent high priority.

Algorithm:

1: Start the process

2: Accept the number of processes in the ready Queue

3: For each process in the ready Q, assign the process id and accept the CPU burst time

4: Sort the ready queue according to the priority number.

5: Set the waiting of the first process as '0' and its burst time as its turnaround time

6: For each process in the Ready Q calculate

(e) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

(f) Turnaround time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

7: Calculate

(g) Average waiting time = Total waiting Time / Number of process

(h) Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the Process

Code:

```
#include<stdio.h>
void main()
{
   int wt[20]={0},bt[20]={0},tat[20]={0},prty[20]={0},i,swt=0,stat=0,max,j,n,m,p[20],x;
   float awt,atat;
printf("Enter  the no. of processes:");
```

```
scanf("%d",&n);
printf("Enter the burst times:");
   for(i=0;i<n;i++)
   {
         scanf("%d",&bt[i]);
   }
printf("\nEnter the priorities:");
   for(i=0;i<n;i++)
   {
         scanf("%d",&prty[i]);
   }
printf("Enter the processes number:");
   for(i=0;i<n;i++)
   {
         scanf("%d",&p[i]);
   }
   for(i=0;i<n;i++)
   {
      for(j=i+1;j<n;j++)
      {

         if(prty[i]>prty[j])
         {
            max=prty[i];
            prty[i]=prty[j];
            prty[j]=max;
            m=p[i];
            p[i]=p[j];
```

```
            p[j]=m;
            x=bt[i];
            bt[i]=bt[j];
            bt[j]=x;
        }
      }
   }
wt[0]=0;
   for(i=1;i<n;i++)
   {
        wt[i]=bt[i-1]+wt[i-1];
   }
   for(i=0;i<n;i++)
   {
        tat[i]=wt[i]+bt[i];
   }
   for(i=1;i<n;i++)
   {
        wt[i]=bt[i-1]+wt[i-1];
        tat[i]=wt[i]+bt[i];
   }
        for(i=0;i<n;i++)
   {
        swt=swt+wt[i];
        stat=stat+tat[i];
   }
awt=swt/n;
atat=stat/n;
```

```
printf("Process\tBT\tWT\tTAT");
printf("\n---------------------------------------");
   for(i=0;i<n;i++)
   {
         printf("\n P%d  \t%d\t%d \t%d",p[i],bt[i],wt[i],tat[i]);
   }
printf("\n---------------------------------------");
printf("\n  \t(AWT)%f   %f(ATAT)\n",awt,atat);
printf("\n---------------------------------------");
}
```

Output:

```
Enter the no of processes:5
Enter the burst times:10
1
2
1
5

Enter the priorities:3
1
4
5
2
Enter the processes number:1
2
3
4
5
```

```
Process BT      WT      TAT
---------------------------------------
 P2     1       0       1
 P5     5       1       6
 P1     10      6       16
 P3     2       16      18
 P4     1       18      19
---------------------------------------
       (AWT)8.000000   12.000000(ATAT)
---------------------------------------

---------------------------------------
Process exited after 37.98 seconds with return value 40
Press any key to continue . . .
```