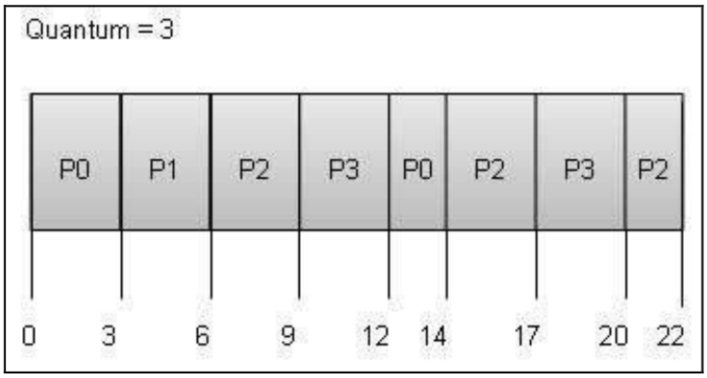
3.C) Write C program to simulate Round Robin CPU scheduling algorithm.

AIM:**To write a C program to implement Round Robin CPU scheduling algorithm.**

DESCRIPTION

- 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.



Wait time of each process is as follows –

Process	**Wait Time : Service Time - Arrival Time**
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

ALGORITHM:

Step 1: Start the program.

Step 2: Initialize all the structure elements.

Step 3: Receive inputs from the user to fill process id, burst time and arrival time.

Step 4: Calculate the waiting time for all the process id.

```
   i.   The waiting time for first instance of a process is calculated as: a[i].waittime=count +
        a[i].arrivt.
  ii.   The waiting time for the rest of the instances of the process is calculated as:
          a) If the time quantum is greater than the remaining burst time then waiting time is
             calculated as: a[i].waittime=count + tq.
          b) Else if the time quantum is greater than the remaining burst time then waiting time is
             calculated as: a[i].waittime=count - remaining burst time
```

Step 5: Calculate the average waiting time and average turnaround time

Step 6: Print the results of the step 4.

Step 7: Stop the program.

PROGRAM :

SOURCE CODE:

```
/* A program to simulate the Round Robin CPU scheduling algorithm */
#include<stdio.h>
struct process
{
        int burst,wait,comp,f;
}p[20]={0,0};
int main()
{
        int n,i,j,totalwait=0,totalturn=0,quantum,flag=1,time=0;
        printf("\nEnter The No Of Process     :");
        scanf("%d",&n);
        printf("\nEnter The Quantum time (in ms) :");
        scanf("%d",&quantum);
        for(i=0;i<n;i++)
        {
                printf("Enter The Burst Time (in ms) For Process #%2d :",i+1);
                scanf("%d",&p[i].burst);
        p[i].f=1;
}
printf("\nOrder Of Execution \n");
printf("\nProcess Starting Ending Remaining");
printf("\n\t\tTime \tTime \t Time");
while(flag==1)
{
        flag=0;
        for(i=0;i<n;i++)
        {
                if(p[i].f==1)
                {
                        flag=1;
                        j=quantum;
                        if((p[i].burst-p[i].comp)>quantum)
                        {
                                p[i].comp+=quantum;
                        }
                        else
                        {
                                p[i].wait=time-p[i].comp;
                                j=p[i].burst-p[i].comp;
                                p[i].comp=p[i].burst;
                                p[i].f=0;
                        }
                        printf("\nprocess # %-3d %-10d %-10d %-10d", i+1, time, time+j,
                p[i].burst-p[i].comp);
                        time+=j;
                }
        }
}
printf("\n\n------------------");
printf("\nProcess \t Waiting Time TurnAround Time ");
for(i=0;i<n;i++)
{
```

```
                printf("\nProcess # %-12d%-15d%-15d",i+1,p[i].wait,p[i].wait+p[i].burst);
                totalwait=totalwait+p[i].wait;
                totalturn=totalturn+p[i].wait+p[i].burst;
        }
        printf("\n\nAverage\n------------------              ");
        printf("\nWaiting  Time: %fms",totalwait/(float)n);
        printf("\nTurnAround Time : %fms\n\n",totalturn/(float)n);
        return 0;
}
```

OUTPUT:

```
$ vi rr.c
$ cc rr.c
$ ./a.out
Enter The No Of Process: 3
Enter The Quantum time (in ms): 5
Enter The Burst Time (in ms) For Process # 1: 25
Enter The Burst Time (in ms) For Process # 2: 30
Enter The Burst Time (in ms) For Process # 3: 54
Order Of Execution
Process             Starting      Ending        Remaining
                    Time          Time          Time
process # 1         0             5             20
process # 2         5             10            25
process # 3         10            15            49
process # 1         15            20            15
process # 2         20            25            20
process # 3         25            30            44
process # 1         30            35            10
process # 2         35            40            15
process # 3         40            45            39
process # 1         45            50            5
process # 2         50            55            10
process # 3         55            60            34
process # 1         60            65            0
process # 2         65          70          5
process # 3         70          75          29
process # 2         75          80          0
process # 3         80          85          24
process # 3         85          90          19
process # 3         90          95          14
process # 3         95          100         9
process # 3         100         105         4
process # 3         105          109         0
------------------
Process             Waiting Time        TurnAround Time
Process # 1               40                  65
Process # 2               50                  80
Process # 3               55                  109
Average
------------------
Waiting  Time: 48.333333ms
TurnAround Time: 84.666667ms
```

RESULT:

Thus the program was executed and verified successfully.

3.D) Write C program to simulate Priority CPU scheduling algorithm.
AIM:To write a C program to implement Priority CPU scheduling algorithm.

Priority Scheduling is a CPU scheduling algorithm in which the CPU performs the task having higher priority at first. If two processes have the same priority then scheduling is done on **FCFS** basis (first come first serve).
Priority Scheduling is of two types : **Preemptive** and **Non-Preemptive**.
Preemptive: In this case, resources can be voluntarily snatched.
Non-Preemptive: In this type, if a process is once started, it will execute completely i.e resources cannot be snatched.
Following are the basic terminologies:
Waiting Time: Time for which the process has to wait in the ready queue.
Turn Around Time: Total time taken by the process for execution (waiting time + burst time).
Example:
Following is the example of non preemptive scheduling with arrival time zero.

Process	**Burst Time**	**Priority**
P1	5	1
P2	7	6
P3	2	4
P4	3	5

Since scheduling is non preemptive, which means that the process will be fully executed once its execution is started. So processes will be executed in the same order of priority.

Order: P2, P4, P3, P1
P2 will be executed from 0 to 7.
P4 will be executed from 7 to 10.
P3 will be executed from 10 to 12.
P1 will be executed from 12 to 17.

Process Id	**Burst Time**	**Wait Time**	**Turn Around Time**
P2	7	0	7
P4	3	7	10
P3	2	10	12
P1	5	12	17

ALGORITHM:

Step 1: Start the Program.
Step 2: Input the number of processes.
Step 3: Input the burst time and priority for each process.
Step 4: Sort the element on the basis of priority.
Step 5: Print order of execution of their process with their time stamp (wait time and turnaround time).
Step 6: End the Program.

Program/Source Code

```
/*
 * C program to implement priority scheduling
 */


#include <stdio.h>


//Function to swap two variables
void swap(int *a,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}
int main()
{
    int n;
    printf("Enter Number of Processes: ");
    scanf("%d",&n);

    // b is array for burst time, p for priority and index for process id
    int b[n],p[n],index[n];
    for(int i=0;i<n;i++)
    {
        printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
        scanf("%d %d",&b[i],&p[i]);
    index[i]=i+1;
}
for(int i=0;i<n;i++)
{
    int a=p[i],m=i;

    //Finding out highest priority element and placing it at its desired position
    for(int j=i;j<n;j++)
    {
        if(p[j] > a)
        {
            a=p[j];
            m=j;
        }
    }

    //Swapping processes
    swap(&p[i], &p[m]);
    swap(&b[i], &b[m]);
    swap(&index[i],&index[m]);
}

// T stores the starting time of process
int t=0;

//Printing scheduled process
printf("Order of process Execution is\n");
for(int i=0;i<n;i++)
{
    printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);
    t+=b[i];
    }
    printf("\n");
    printf("Process Id     Burst Time   Wait Time    TurnAround Time\n");
    int wait_time=0;
    for(int i=0;i<n;i++)
    {
        printf("P%d          %d          %d          %d\n",index[i],b[i],wait_time,wait_time + b[i]);
        wait_time += b[i];
    }
    return 0;
}
```

Run Time Testcases

In this case, we enter “3” as the number of processes, and the burst time and priority value are “p1: 10 2”, “p2: 5 0”, and “p3: 8 1”.

```
Enter Number of Processes: 3
Enter Burst Time and Priority Value for Process 1: 10 2
Enter Burst Time and Priority Value for Process 2: 5 0
Enter Burst Time and Priority Value for Process 3: 8 1
Order of process Execution is
P1 is executed from 0 to 10
P3 is executed from 10 to 18
P2 is executed from 18 to 23


Process Id     Burst Time   Wait Time    TurnAround Time
    P1           10          0            10
    P3           8           10           18
    P2           5           18           23
```