Assignment 4

221071

AIM: Write a program to show the demonstration of Scheduling Algorithms: 1) FCFS 2) SJF 3) Round Robin

2) THEORY:

-) What is CPU **Scheduling**?

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

-) Why do we need scheduling?
A typical process involves both I/O time and CPU time. In a uni programming system like MS-DOS, time spent waiting for I/O is wasted and CPU is free during this time. In multi programming systems, one process can use CPU while another is waiting for I/O. This is possible only with process scheduling.

Non-Preemptive Scheduling

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.
It is the only method that can be used on certain hardware platforms, because It does not require the special hardware(for example: a timer) needed for preemptive scheduling.

Preemptive Scheduling

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

CPU Scheduling: Scheduling Criteria

There are many different criterias to check when considering the **"best"** scheduling algorithm, they are:

1) CPU Utilization

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

2) Throughput

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

3) Turnaround Time

It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

4) Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

5) Load Average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

6) Response Time

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Scheduling Algorithms

1) First Come First Serve (FCFS)

First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue. So, when CPU becomes free, it should be assigned to the process at the beginning of the queue.

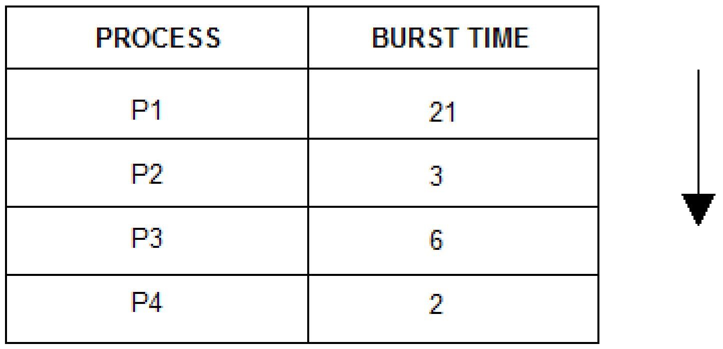
Characteristics of FCFS method:

- It offers non-preemptive and pre-emptive scheduling algorithm.

•Jobs are always executed on a first-come, first-serve basis
•It is easy to implement and use.
•However, this method is poor in performance, and the general wait time is quite high.

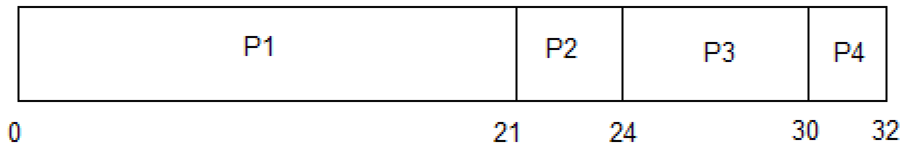
Example

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with **Arrival Time** 0, and given **Burst Time**, let's find the average waiting time using the FCFS scheduling algorithm.



PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

The average waiting time will be 18.75 ms

For the above given proccesses, first **P1** will be provided with the CPU resources,

•Hence, waiting time for **P1** will be 0

•**P1** requires 21 ms for completion, hence waiting time for **P2** will be 21 ms

•Similarly, waiting time for process **P3** will be execution time of **P1** + execution time for **P2**, which will be (21 + 3) ms = 24 ms.

•For process **P4** it will be the sum of execution times of **P1**, **P2** and **P3**.

The **GANTT chart** above perfectly represents the waiting time for each process.

2)Shortest-Job-First (SJF) Scheduling

SJF is a full form of (Shortest job first) is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

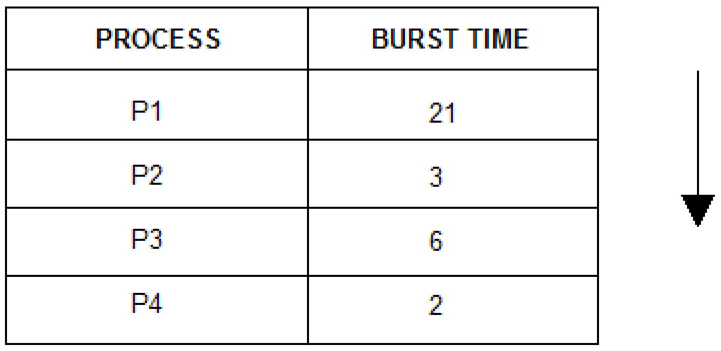
Characteristics of SJF Scheduling

- •It is associated with each job as a unit of time to complete.
- •In this method, when the CPU is available, the next process or job with the shortest completion time will be executed first.
- •It is Implemented with non-preemptive policy.
- •This algorithm method is useful for batch-type processing, where waiting for jobs to complete is not critical.
- •It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

Example

Non Pre-emptive Shortest Job First

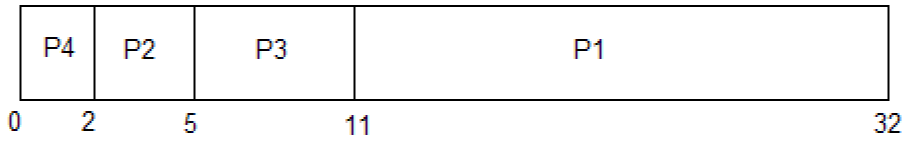
Consider the below processes available in the ready queue for execution, with **arrival time** as 0 for all and given **burst times**.



PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



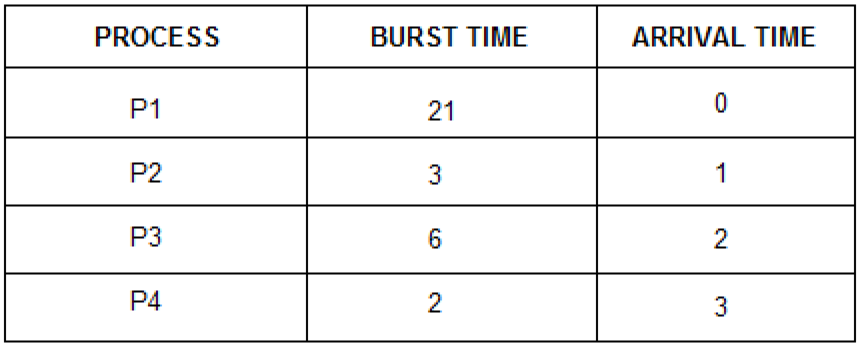
Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

As you can see in the **GANTT chart** above, the process **P4** will be picked up first as it has the shortest burst time, then **P2**, followed by **P3** and at last **P1**.

We scheduled the same set of processes using the First come first serve algorithm in the previous tutorial, and got average waiting time to be 18.75 ms, whereas with SJF, the average waiting time comes out 4.5 ms.

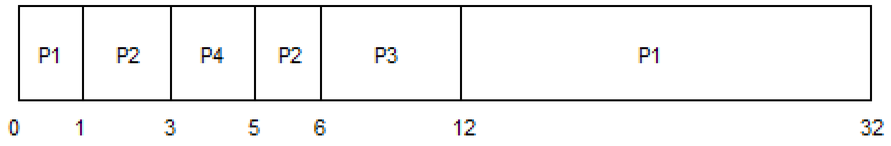
Pre-emptive Shortest Job First

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with **short burst time** arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.



PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



The average waiting time will be, ((5-3) + (6-2) + (12-1))/4 = 4.25 ms

The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

As you can see in the **GANTT chart** above, as **P1** arrives first, hence it's execution starts immediately, but just after 1 ms, process **P2** arrives with a **burst time** of 3 ms which is less than the burst time of **P1**, hence the process **P1**(1 ms done, 20 ms left) is preemptied and process **P2** is executed.

As **P2** is getting executed, after 1 ms, **P3** arrives, but it has a burst time greater than that of **P2**, hence execution of **P2** continues. But after another millisecond, **P4** arrives with a burst time of 2 ms, as a result **P2**(2 ms done, 1 ms left) is preemptied and **P4** is executed.

After the completion of **P4**, process **P2** is picked up and finishes, then **P2** will get executed and at last **P1**.

The Pre-emptive SJF is also known as **Shortest Remaining Time First**, because at any given point of time, the job with the shortest remaining time is executed first.

3)Round Robin Scheduling

Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation free execution of processes.

Characteristics of Round-Robin Scheduling

- Round robin is a hybrid model which is clock-driven
- Time slice should be minimum, which is assigned for a specific task to be processed. However, it may vary for different processes.
- It is a real time system which responds to the event within a specific time limit.

Example

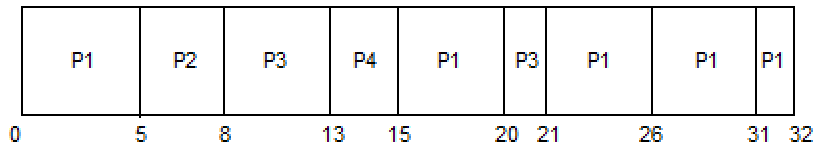
A fixed time is allotted to each process, called quantum, for execution.

Once a process is executed for given time period that process is preemptied and other process executes for given time period.

Context switching is used to save states of preemptied processes.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,



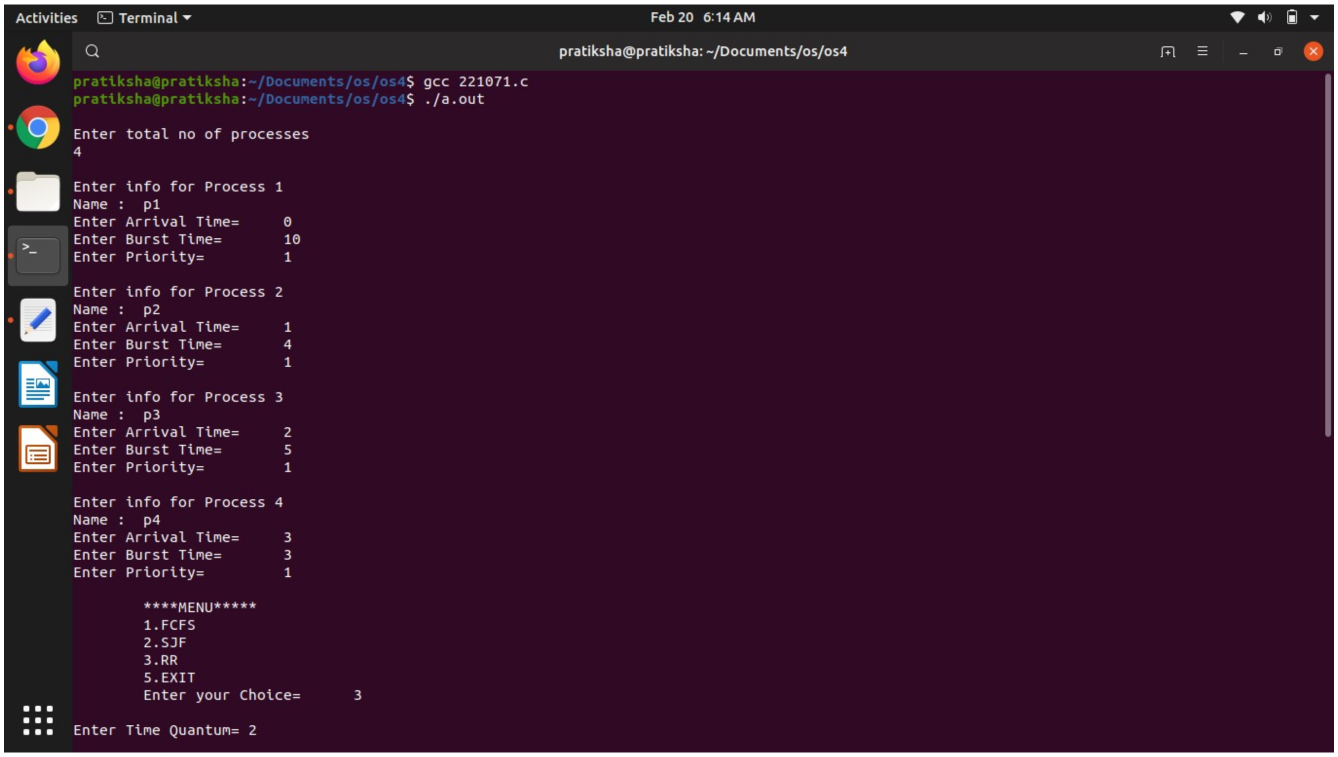
The average waiting time will be, 11 ms.

3) code:

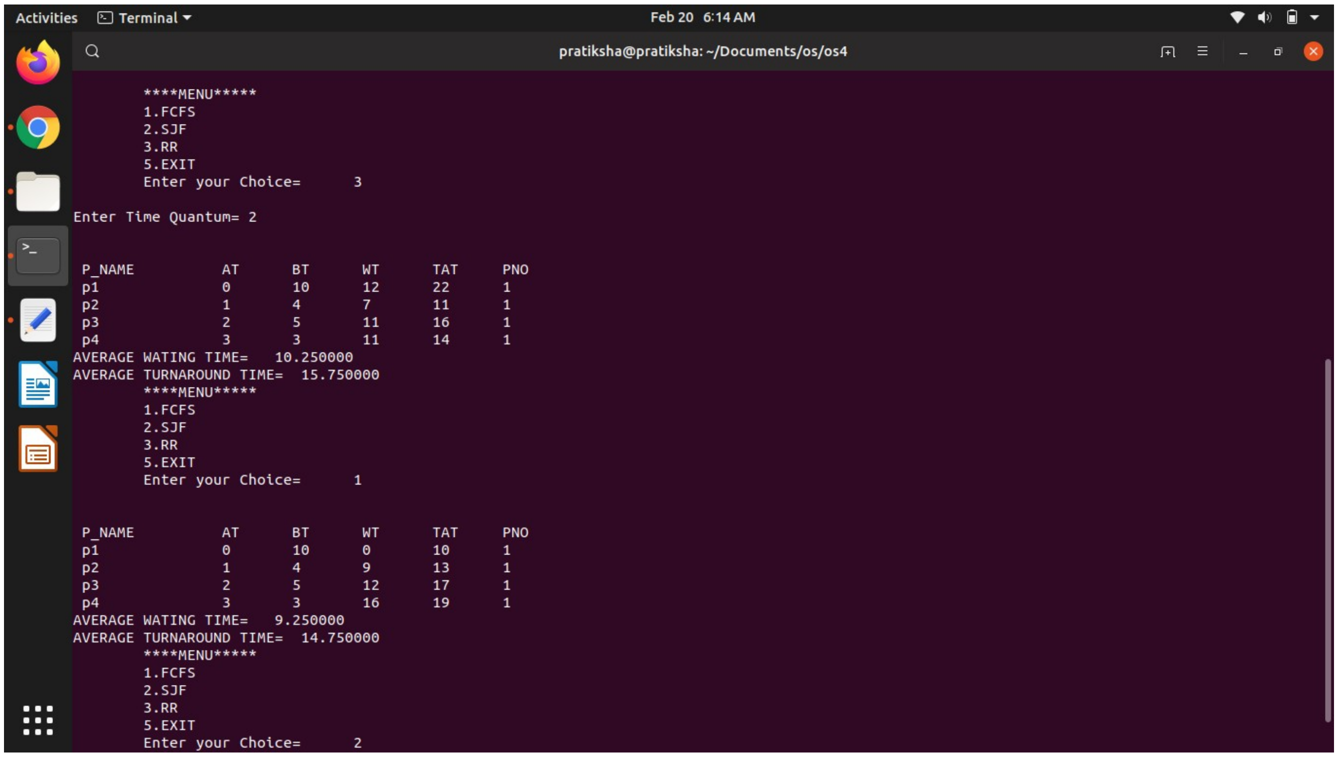
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 100
typedef struct process
{
   int BT,AT,TAT,WT,PNO,PID;
   char name[10];
}process;
typedef struct RQ
{
   process pr[MAX];
   int f,r;
}RQ;

void get_info(process [],int *);
void FCFS(process [],int);
void at_sort(process [],int);
void bt_sort(RQ []);
void disp_table(process [],int );
void SJF(process [],int);
void RR(process p[],int n,int);
float cal_avgwt(process [],int );
float cal_avgtat(process [],int );
void menu()
{
    printf("\n\t****MENU*****");
    printf("\n\t1.FCFS");
    printf("\n\t2.SJF");
    printf("\n\t3.RR");
    printf("\n\t5.EXIT");
    printf("\n\tEnter your Choice= \t");
}
int main()
{
    int ch,TQ,n;
    process P[MAX];
    float avg_WT,avg_TAT;


    get_info(P,&n);
    do
    {

        menu();
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
            {
                at_sort(P,n);
                FCFS(P,n);
                disp_table(P,n);
                avg_WT=cal_avgwt(P,n);
                avg_TAT=cal_avgtat(P,n);
                printf("\nAVERAGE WATING TIME=   %f",avg_WT);
                printf("\nAVERAGE TURNAROUND TIME=  %f",avg_TAT);
                break;
            }
            case 2:
            {
                SJF(P,n);
                disp_table(P,n);
                avg_WT=cal_avgwt(P,n);
                avg_TAT=cal_avgtat(P,n);
                printf("\nAVERAGE WATING TIME=   %f",avg_WT);
                printf("\nAVERAGE TURNAROUND TIME=  %f",avg_TAT);
                break;
            }
            case 3:
            {
                printf("\nEnter Time Quantum= ");
                        scanf("%d",&TQ);
                RR(P,n,TQ);
                disp_table(P,n);
                avg_WT=cal_avgwt(P,n);
                avg_TAT=cal_avgtat(P,n);
                printf("\nAVERAGE WATING TIME=   %f",avg_WT);
                printf("\nAVERAGE TURNAROUND TIME=  %f",avg_TAT);
                break;
            }
            case 4:
                break;
        }

    }while(ch!=5);
     return 0;
}


float cal_avgwt(process p[],int n)
{
    float avg=0;
    int i;
    for(i=0;i<n;i++)
    {
        avg+=p[i].WT;
    }
    avg=avg/n;
    return avg;
}
float cal_avgtat(process p[],int n)
{
    float avg=0;
    int i;
    for(i=0;i<n;i++)
    {
        avg+=p[i].TAT;
    }
    avg=avg/n;
    return avg;
}
int get_first_process(process p[],int n)
{
    int min,j,in;
    min=p[0].AT;
    for(j=0;j<n;j++)
    {
        if(p[j].AT<min)
        {
            in=j;
        }
       }
       return in;
}
void check_arrival(RQ *r,process p[],int time,int n)
{
    int i,j,flag=0;
    for(i=0;i<n;i++)
    {
        for(j=0;j<=r->r;j++)
        {
            if(strcmp(p[i].name,r->pr[j].name)==0)
                flag=1;
            else
                break;
        }
        if(p[i].AT == time && flag==0)
        {
            r->r=r->r + 1;
            r->pr[r->r]=p[i];
        }
        flag=0;
    }
}
void RR(process p[],int n,int tq)
{
    int count=0,i,start,time=0;
    RQ r;
    r.f=r.r=-1;
    start=get_first_process(p,n);
    r.pr[0]=p[start];
    r.f=r.r=0;
    check_arrival(&r,p,time,n);
    while(r.f!=-1)
    {
        for(count=0;count<tq;count++)
        {
            r.pr[r.f].BT--;
            time++;
            if(r.pr[r.f].BT==0)
                break;
            check_arrival(&r,p,time,n);
        }
        if(r.pr[r.f].BT!=0)
        {
            r.pr[r.r+1]=r.pr[r.f];
            r.r++;
        }
        else
        {
            p[r.pr[r.f].PID].TAT=time - r.pr[r.f].AT;
            p[r.pr[r.f].PID].WT=p[r.pr[r.f].PID].TAT - p[r.pr[r.f].PID].BT;
        }
        if(r.f==r.r)
            r.f=r.r=-1;
        else
            r.f++;
    }
}
void bt_sort(RQ *r)
{
    int i,j;
    process temp;
    for(i=r->f;i<=r->r;i++)
    {
        for(j=i+1;j<=r->r;j++)
        {
            if(r->pr[j].BT<r->pr[i].BT)
            {
                temp=r->pr[j];
                r->pr[j]=r->pr[i];
                r->pr[i]=temp;
            }
        }
    }
}

void SJF(process p[],int n)
{
    int time=0,start;
    RQ r;
    r.f=r.r=-1;
    start=get_first_process(p,n);
    r.pr[0]=p[start];
    //p[start].WT=0;
    //p[start].TAT=p[start].BT;
    r.f=r.r=0;
    check_arrival(&r,p,time,n);
    while(r.f!=-1)
    {
        p[r.pr[r.f].PID].WT=time - r.pr[r.f].AT;
        p[r.pr[r.f].PID].TAT=p[r.pr[r.f].PID].WT + r.pr[r.f].BT;
        while(r.pr[r.f].BT != 0)
        {
            time++;
            check_arrival(&r,p,time,n);
            r.pr[r.f].BT--;
        }
        if(r.f==r.r)
            r.f=r.r=-1;
        else
            r.f++;
        bt_sort(&r);
    }
}
int get_total_time(process p[],int n)
{
    int i,sum=0;
    for(i=0;i<n;i++)
    {
        sum+=p[i].BT;
    }
    return sum;
}
void at_sort(process p[],int n)
{
    int i,j;
    process temp;
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(p[j].AT<p[i].AT)
            {
                temp=p[j];
                p[j]=p[i];
                p[i]=temp;
            }
        }
    }
}
void disp_table(process p[],int n)
{
    int i;
    printf("\n\n P_NAME \t AT \t BT \t WT \t TAT \t PNO");
    for(i=0;i<n;i++)
    {
        printf("\n %-10s \t %d \t %d \t %d \t %d \t
%d",p[i].name,p[i].AT,p[i].BT,p[i].WT,p[i].TAT,p[i].PNO);
    }
}
```

```
void get_info(process p[],int *n)
{
    int i;
    printf("\nEnter total no of processes\n");
    scanf("%d",n);
    for(i=0;i<*n;i++)
    {
        printf("\nEnter info for Process %d",i+1);
        printf("\nName :\t");
        scanf("%s",p[i].name);
        printf("Enter Arrival Time= \t");
        scanf("%d",&p[i].AT);
        printf("Enter Burst Time= \t");
        scanf("%d",&p[i].BT);
        printf("Enter Priority= \t");
        scanf("%d",&p[i].PNO);
        p[i].PID=i;
    }
}
void FCFS(process p[],int n)
{
    int i,j,sum=0;
    p[0].TAT = p[0].BT;
    p[0].WT = 0;
    sum = p[0].BT;
    for(i=1;i<n;i++)
    {
        p[i].WT=sum - p[i].AT;
        p[i].TAT=p[i].WT + p[i].BT;
        sum+=p[i].BT;
    }
}
```

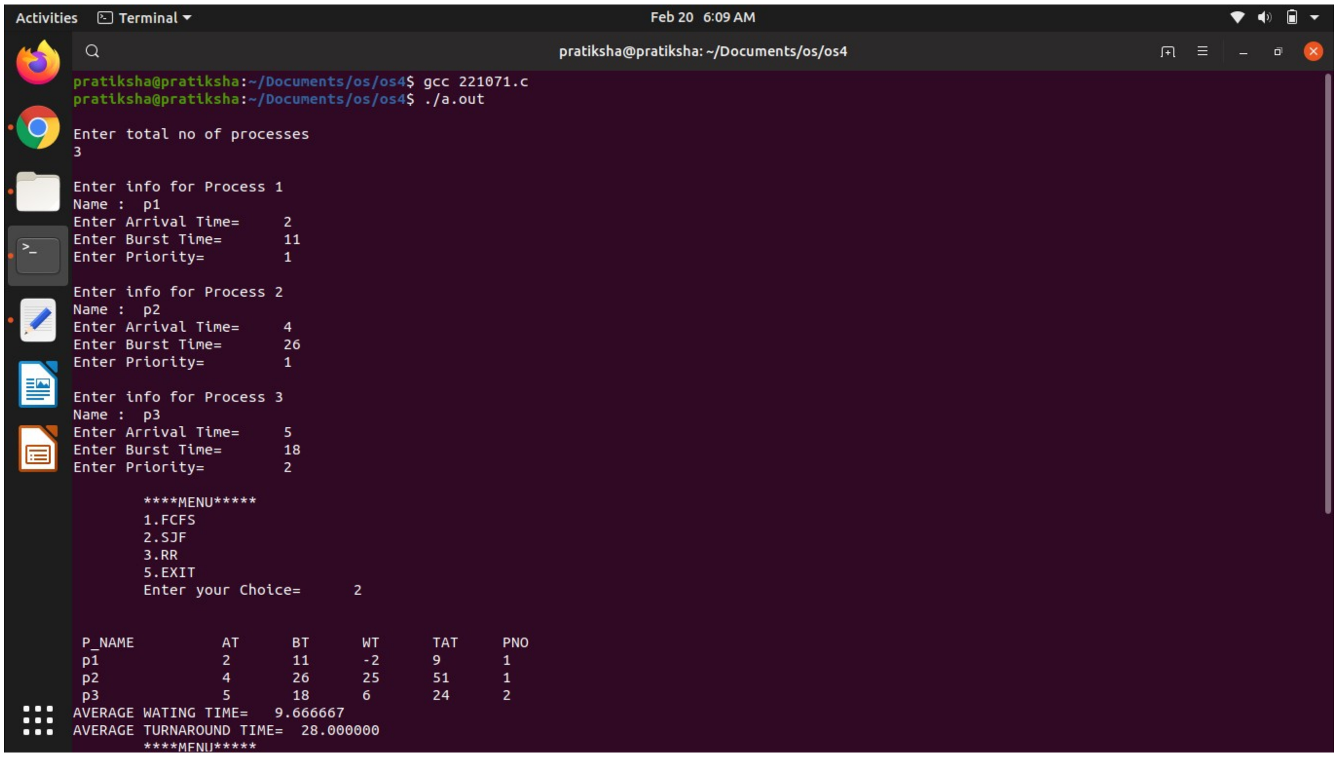
4) output.



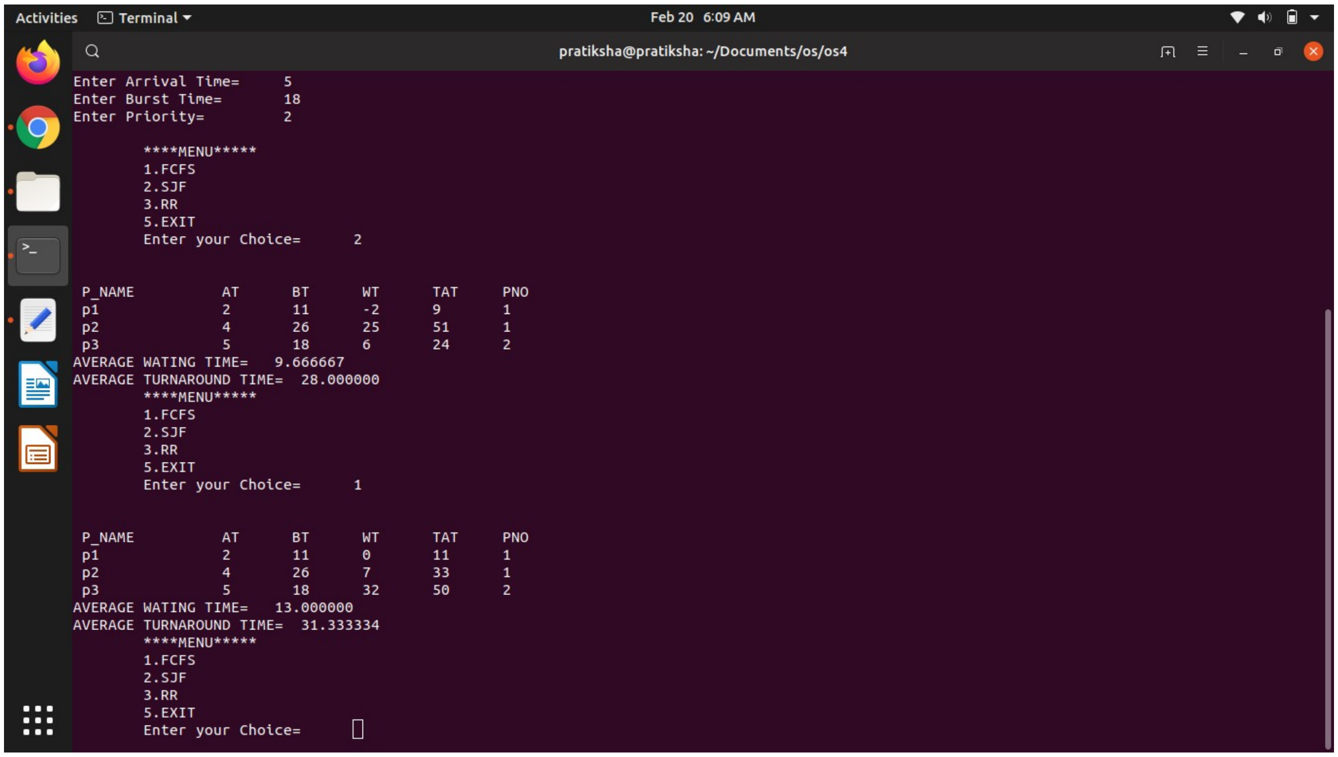
```
pratiksha@pratiksha:~/Documents/os/os4$ gcc 221071.c
pratiksha@pratiksha:~/Documents/os/os4$ ./a.out

Enter total no of processes
4

Enter info for Process 1
Name :  p1
Enter Arrival Time=     0
Enter Burst Time=       10
Enter Priority=         1

Enter info for Process 2
Name :  p2
Enter Arrival Time=     1
Enter Burst Time=       4
Enter Priority=         1

Enter info for Process 3
Name :  p3
Enter Arrival Time=     2
Enter Burst Time=       5
Enter Priority=         1

Enter info for Process 4
Name :  p4
Enter Arrival Time=     3
Enter Burst Time=       3
Enter Priority=         1

        ****MENU*****
        1.FCFS
        2.SJF
        3.RR
        5.EXIT
        Enter your Choice=      3

Enter Time Quantum= 2
```



```
        ****MENU*****
        1.FCFS
        2.SJF
        3.RR
        5.EXIT
        Enter your Choice=      3

Enter Time Quantum= 2


 P_NAME          AT      BT      WT      TAT     PNO
 p1              0       10      12      22      1
 p2              1       4       7       11      1
 p3              2       5       11      16      1
 p4              3       3       11      14      1
AVERAGE WATING TIME=    10.250000
AVERAGE TURNAROUND TIME=  15.750000
        ****MENU*****
        1.FCFS
        2.SJF
        3.RR
        5.EXIT
        Enter your Choice=      1


 P_NAME          AT      BT      WT      TAT     PNO
 p1              0       10      0       10      1
 p2              1       4       9       13      1
 p3              2       5       12      17      1
 p4              3       3       16      19      1
AVERAGE WATING TIME=    9.250000
AVERAGE TURNAROUND TIME=  14.750000
        ****MENU*****
        1.FCFS
        2.SJF
        3.RR
        5.EXIT
        Enter your Choice=      2
```



```
pratiksha@pratiksha:~/Documents/os/os4$ gcc 221071.c
pratiksha@pratiksha:~/Documents/os/os4$ ./a.out

Enter total no of processes
3

Enter info for Process 1
Name :  p1
Enter Arrival Time=     2
Enter Burst Time=       11
Enter Priority=         1

Enter info for Process 2
Name :  p2
Enter Arrival Time=     4
Enter Burst Time=       26
Enter Priority=         1

Enter info for Process 3
Name :  p3
Enter Arrival Time=     5
Enter Burst Time=       18
Enter Priority=         2

        ****MENU*****
        1.FCFS
        2.SJF
        3.RR
        5.EXIT
        Enter your Choice=      2


 P_NAME          AT      BT      WT      TAT     PNO
 p1              2       11      -2      9       1
 p2              4       26      25      51      1
 p3              5       18      6       24      2
AVERAGE WATING TIME=    9.666667
AVERAGE TURNAROUND TIME=  28.000000
```



```
Enter Arrival Time=     5
Enter Burst Time=       18
Enter Priority=         2

        ****MENU*****
        1.FCFS
        2.SJF
        3.RR
        5.EXIT
        Enter your Choice=      2


 P_NAME          AT      BT      WT      TAT     PNO
 p1              2       11      -2      9       1
 p2              4       26      25      51      1
 p3              5       18      6       24      2
AVERAGE WATING TIME=    9.666667
AVERAGE TURNAROUND TIME=  28.000000
        ****MENU*****
        1.FCFS
        2.SJF
        3.RR
        5.EXIT
        Enter your Choice=      1


 P_NAME          AT      BT      WT      TAT     PNO
 p1              2       11      0       11      1
 p2              4       26      7       33      1
 p3              5       18      32      50      2
AVERAGE WATING TIME=    13.000000
AVERAGE TURNAROUND TIME=  31.333334
        ****MENU*****
        1.FCFS
        2.SJF
        3.RR
        5.EXIT
        Enter your Choice=
```