NAME: Meet Raut

DIV: S21

ROLL.NO: 2201084

Experiment 5:

- **AIM:** **a) To study and implement non preemptive scheduling algorithm FCFS.**

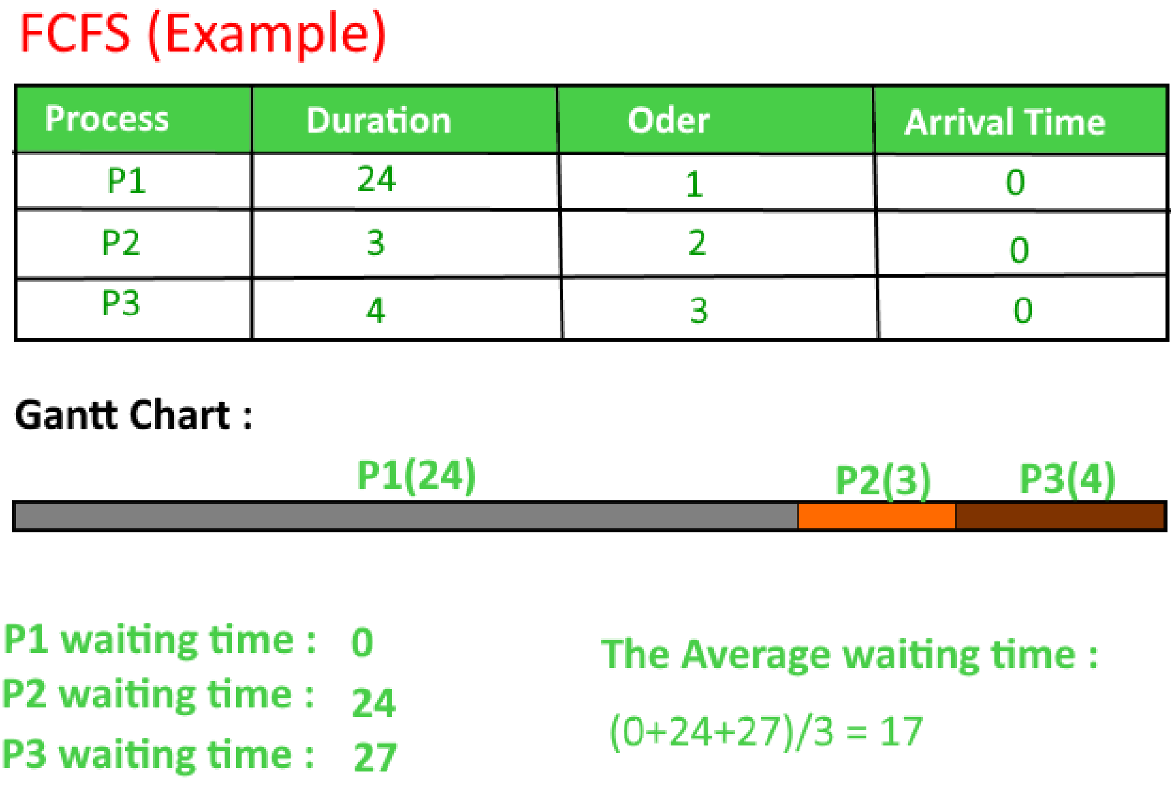
  b) To study and implement preemptive scheduling algorithm SRTF

- **THEORY:**

Non preemptive algorithm : FCFS, SJF

1] FCFS : First Come First Serve

It is a non-preemptive scheduling algorithm and the criteria for this is the arrival time of the process CPU is allotted to the process that requires it first. Jobs arriving later are placed at the end of the queue.



FCFS (Example)

Process	Duration	Oder	Arrival Time
P1	24	1	0
P2	3	2	0
P3	4	3	0

Gantt Chart :

P1 waiting time : 0

P2 waiting time : 24

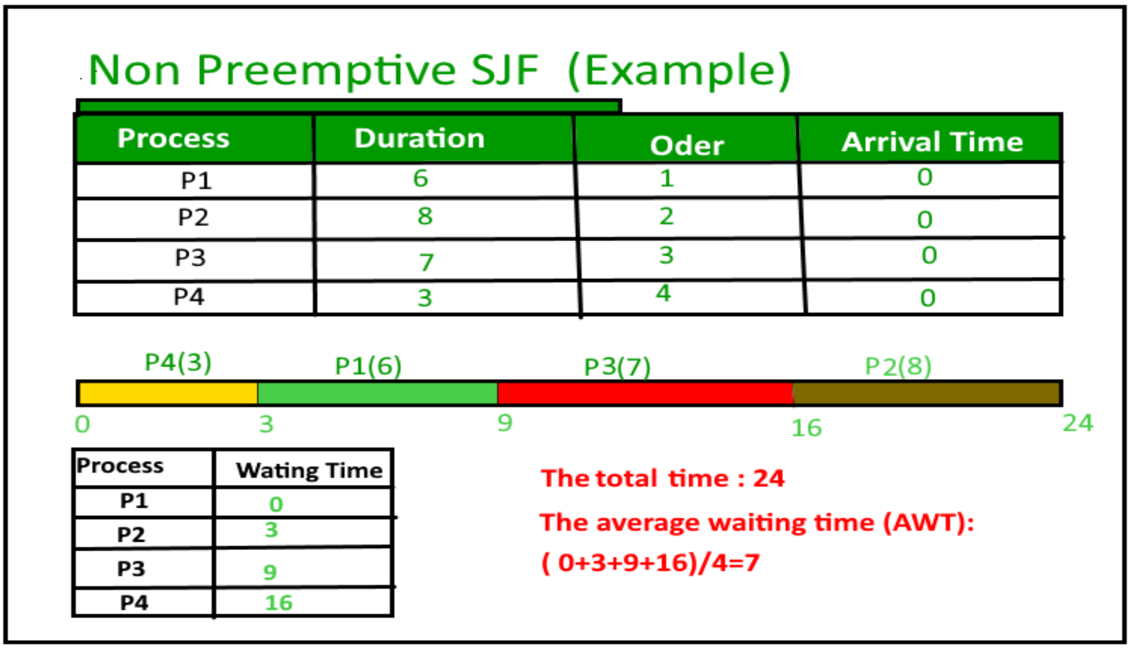
P3 waiting time : 27

The Average waiting time :

(0+24+27)/3 = 17

2] SJF : shortest job first

This is a non preemptive scheduling algorithm, which associates with each process the length of the processes next CPU first.
Criteria : Burst Time.
This algorithm assigns processes according to BT if BT of two processes is the same we use FCFS scheduling criteria.



Non Preemptive SJF (Example)

Process	Duration	Oder	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0

P4(3)
P1(6)
P3(7)
P2(8)
0
3
9
16
24

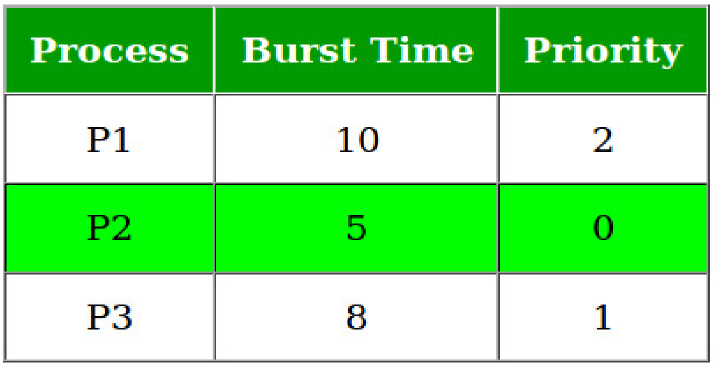
Process	Wating Time
P1	0
P2	3
P3	9
P4	16

The total time : 24

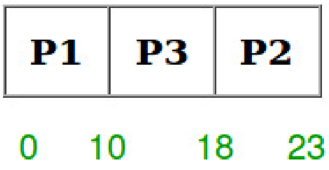
The average waiting time (AWT):
(0+3+9+16)/4=7

3] Priority Scheduling

This is a non preemptive scheduling algorithm. Each process here has a priority that is either assigned already or externally done.



Process	Burst Time	Priority
P1	10	2
P2	5	0
P3	8	1

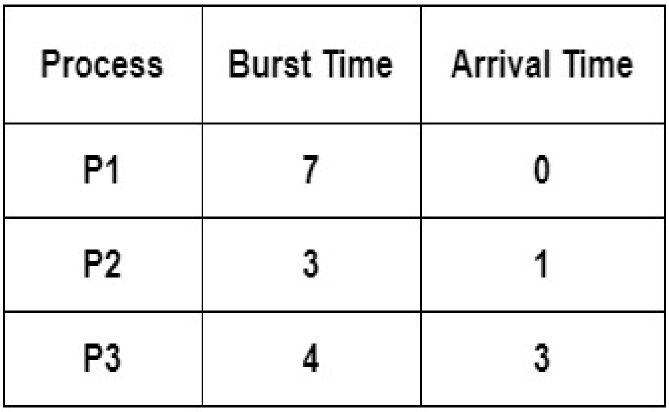


Preemptive Scheduling Algorithm : SRTF/STRN

Shortest Remaining Time First / Shortest Remaining Time Next scheduling.

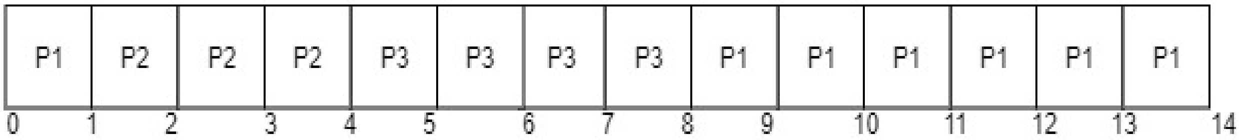
It is a preemptive SJF Algorithm. The choice arrives when a new process arrives as the ready queue. While a previous process is still executing. The next CPU burst if the newly arrived process may be shorter than what is left of the currently executing process. A preemptive SJF will preempt the current executing process and not allow the currently running process to finish its CPU burst.

Round Robin is also one preemptive algorithm.



Process	Burst Time	Arrival Time
P1	7	0
P2	3	1
P3	4	3

The Gantt Chart for SRTF will be:

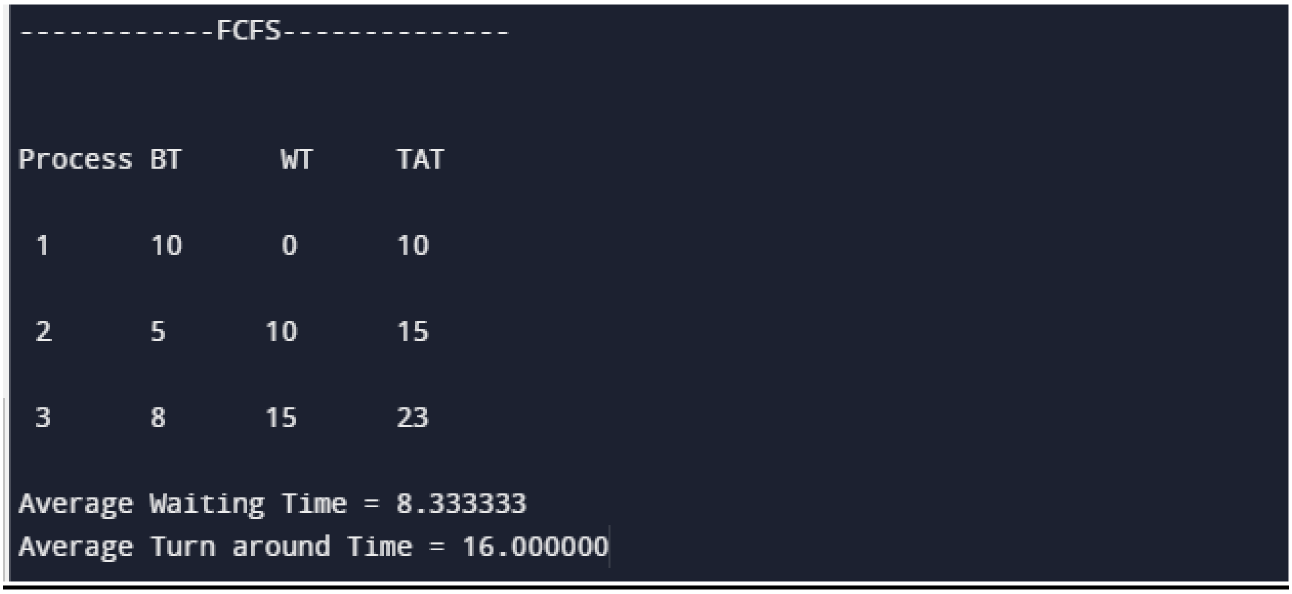


- **C PROGRAM FOR FCFS:**

```
//FCFS SCHEDULING
#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
   wt[0]=0;

   for (int i = 1; i<n; i++)
     wt[i]=bt[i-1] + wt[i-1];
}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])
{
   for (int i=0; i<n; i++)
   tat[i]=bt[i] + wt[i];
}

void findavgTime(int processes[], int n, int bt[])
{
   int wt[n], tat[n], total_wt=0, total_tat=0;
   findWaitingTime(processes, n, bt, wt);
   findTurnAroundTime(processes, n, bt, wt, tat);

   printf("Process BT      WT     TAT\n");

   for(int i=0; i<n; i++)
   {
      total_wt=total_wt+wt[i];
```

```
        total_tat=total_tat+tat[i];
        printf("\n %d ", (i+1));
        printf("     %d ", bt[i]);
        printf("     %d ", wt[i]);
        printf("     %d\n ", tat[i]);
    }
    float s=(float)total_wt / (float)n;
    float t=(float)total_tat / (float)n;
    printf("\n");
    printf("Average Waiting Time = %f",s);
    printf("\n");
    printf("Average Turn around Time = %f",t);

}

int main()
{
    printf("------------FCFS--------------\n\n\n");
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof processes[0];

    int burst_time[]= {10, 5, 8};

    findavgTime(processes, n, burst_time);

    return 0;
}
```

- **<u>OUTPUT:</u>**

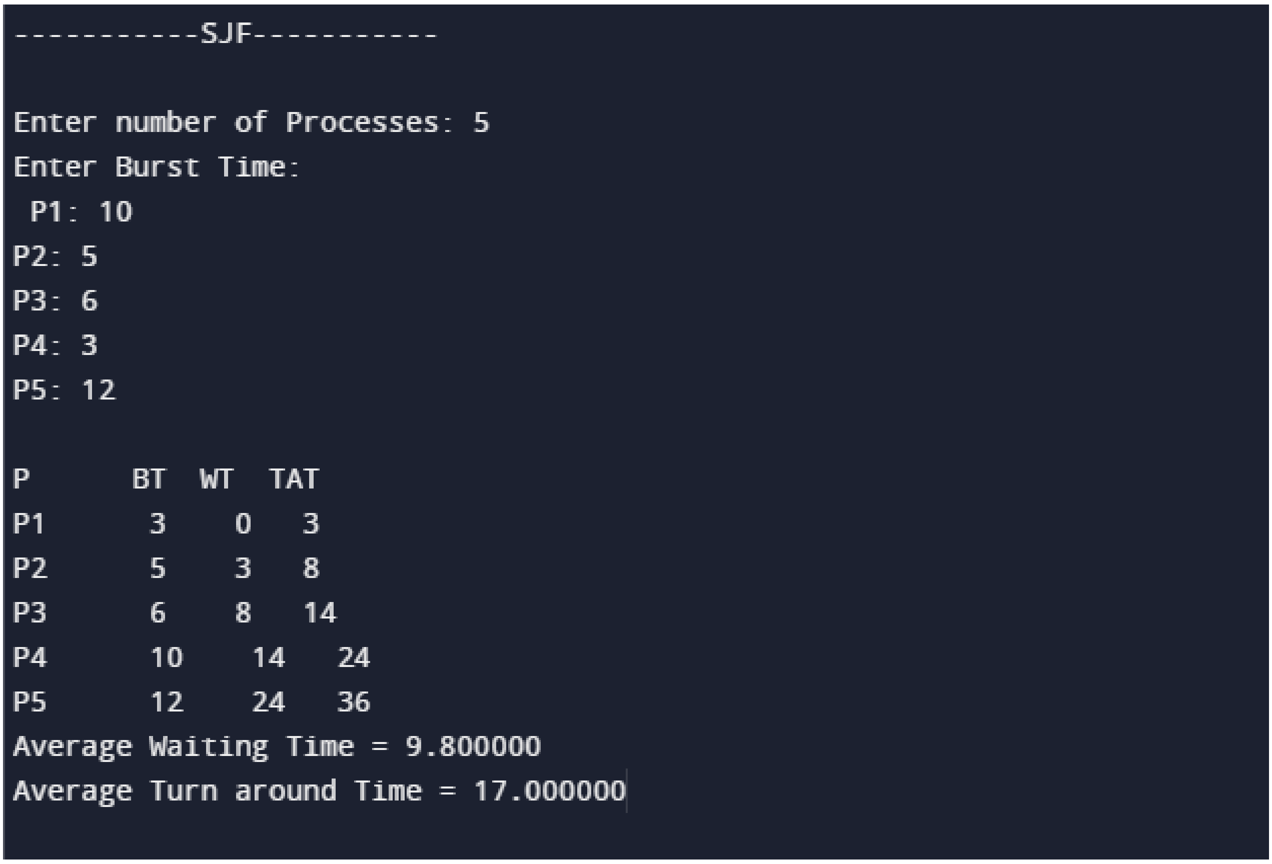


```
------------FCFS--------------


Process BT      WT     TAT

 1      10      0      10

 2      5      10      15

 3      8      15      23

Average Waiting Time = 8.333333
Average Turn around Time = 16.000000
```

- **C PROGRAM FOR NON-PREEMPTIVE SJF:**

```
#include<stdio.h>

int main()
{
   int A[100][4];
   int i, j, n, total = 0, index, temp;
   float avg_wt, avg_tat;

   printf("-----------SJF-----------\n\n");
   printf("Enter number of Processes: ");
   scanf("%d", &n);
   printf("Enter Burst Time:\n ");
   for(i=0; i<n; i++)
   {
      printf("P%d: ", i+1);
      scanf("%d", &A[i][1]);
      A[i][0] = i+1;
   }

   for(i=0; i<n; i++) {
      index=i;
      for(j=i+1; j<n; j++)
      if (A[j][1] < A[index][1])
      index=j;
      temp=A[i][1];
      A[i][1]=A[index][1];
      A[index][1]=temp;
   }

   A[0][2]=0;

   for(i=1; i<n; i++) {
      A[i][2]=0;
      for (j=0; j<i; j++)
      A[i][2]+=A[j][1];
      total+=A[i][2];
   }

   avg_wt=(float)total / n;
   total=0;
   printf("\n");
   printf("P     BT  WT  TAT \n");
   for(i=0; i<n; i++) {
      A[i][3]=A[i][1] + A[i][2];
      total+= A[i][3];
      printf("P%d     %d    %d   %d\n", A[i][0], A[i][1], A[i][2], A[i][3]);
   }
   avg_tat=(float)total / n;
   printf("Average Waiting Time = %f", avg_wt);
   printf("\nAverage Turn around Time = %f", avg_tat);
}
```

- **OUTPUT:**



```
-----------SJF-----------

Enter number of Processes: 5
Enter Burst Time:
 P1: 10
P2: 5
P3: 6
P4: 3
P5: 12

P      BT  WT  TAT
P1      3    0   3
P2      5    3   8
P3      6    8   14
P4      10    14   24
P5      12    24   36
Average Waiting Time = 9.800000
Average Turn around Time = 17.000000
```

CONCLUSION: **Hence, we have successfully implemented pre-emptive and non preemptive scheduling algorithms.**