INTRODUCTION

Operating Systems:

- An operating system is a program that manages a computer's hardware.
- It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware.
- An operating system acts as a resource allocator and it is a control program.

Purpose: The purpose of an operating system is to provide an environment in which user can execute the programs.

Goals:

1. Its primary goal is to make the computer system convenient for use.
2. Its secondary goal is to use the computer hardware in efficient manner.

Examples of Operating Systems are:

Windows, Linux, Unix, Max OS etc.

1. **Write C programs to simulate the following CPU Scheduling algorithms a) FCFS b) SJF c) Round Robin d) priority**
   a) **FCFS**

   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: Set the waiting time of the first process as '0' and its burst time and its turn around time
   5: for each process in the Ready Q calculate
   a. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
   b. Turnaround time for Process(n)= waiting time of Process(n)+ Burst time for process(n)
   6: Calculate
   a. Average waiting time = Total waiting Time / Number of process
   b. Average Turnaround time = Total Turnaround Time / Number of process
   7: Stop the process.

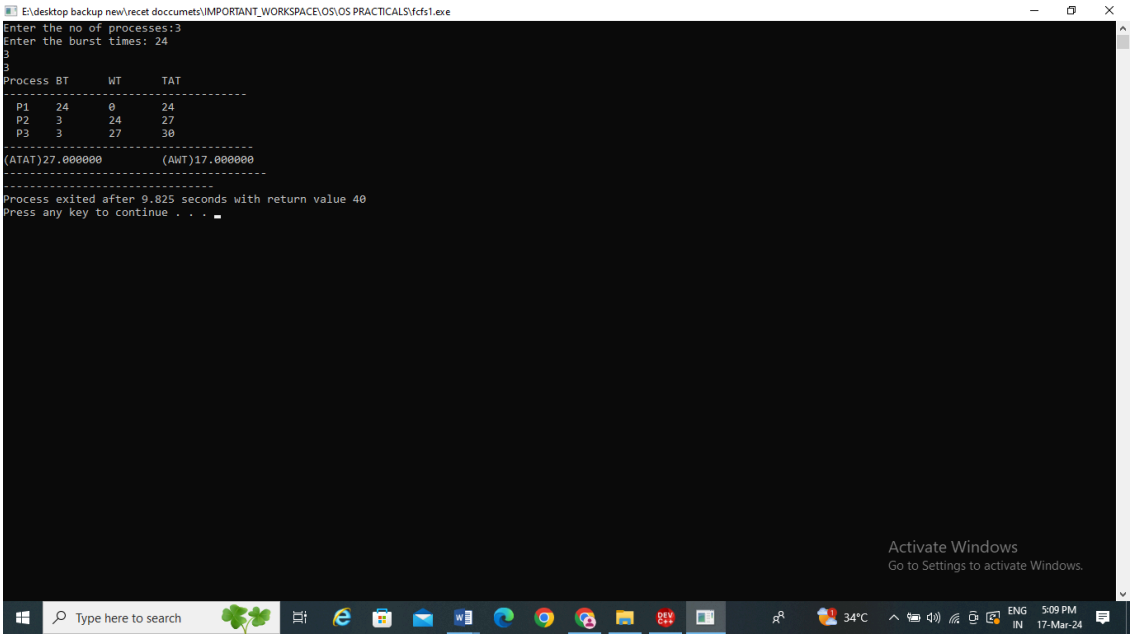
   Code:

```
#include<stdio.h>
void main()
```

```
{
   int wt[20]={0},bt[20]={0},tat[20]={0},n,i,swt=0,stat=0;
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]);
   }
 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=0;i<n;i++)
   {
      swt=swt+wt[i];
   }
      awt=swt/n;

 for(i=0;i<n;i++)
   {
    stat=stat+tat[i];
   }
 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",i+1,bt[i], wt[i], tat[i]);
   }
 printf("\n-----------------------------------");
 printf("\t(ATAT)%f \t(AWT)%f\n",atat,awt);
 printf("-----------------------------------");
 }
```

Output:



b) SJF

Shortest job First: The criteria of this algorithm are which process having the smallest CPU
burst, CPU is assigned to that next process. If two process having the same CPU burst time FCFS is used to break the tie.

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: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst
time.
5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.
6: For each process in the ready queue, calculate
(a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)
7: Calculate
(c) Average waiting time = Total waiting Time / Number of process
(d) Average Turnaround time = Total Turnaround Time / Number of process
8: Stop the process

Code:

```
#include<stdio.h>
void main()
{
```

```
    int wt[20] ={0}, bt[20]={0}, tat[20]={0}, i, swt=0, stat=0, max, j,
n, p[20],m;
    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("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(bt[i]>bt[j])
                    {
                            max=bt[i];
                            m=p[i];
                            bt[i]=bt[j];
                            p[i]=p[j];
                            bt[j]=max;
                            p[j]=m;
                    }
            }
    }
            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=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   %d  \t%d\t%d \t%d",p[i],bt[i], wt[i], tat[i]);
        }
        printf("\n----------------------------------");
        printf("\n       \t(ATAT)%f %f(AWT)\n",atat,awt);
        printf("----------------------------------");
}
```

Output: