**OS ASSIGNMENT NO.4**

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

**THEORY :**

# What is Process Scheduling?

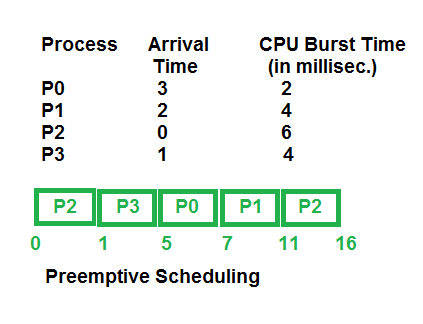
The act of determining which process is in the **ready** state, and should be moved to the **running** state is known as **Process Scheduling**.

The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

Scheduling fell into one of the two general categories:

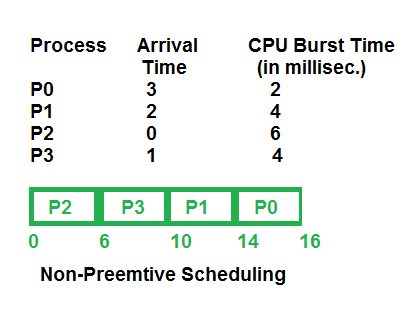
1. Preemptive Scheduling:  
Preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for the limited amount of time and then is taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in ready queue till it gets next chance to execute.

Algorithms based on preemptive scheduling are: [Round Robin (RR)](https://www.geeksforgeeks.org/program-round-robin-scheduling-set-1/),[Shortest Remaining Time First (SRTF)](https://www.geeksforgeeks.org/program-shortest-job-first-scheduling-set-2srtf-make-changesdoneplease-review/), [Priority (preemptive version)](https://www.geeksforgeeks.org/program-for-preemptive-priority-cpu-scheduling/), etc.



2. Non-Preemptive Scheduling:  
Non-preemptive Scheduling is used when a process terminates, or a process switches from running to waiting state. In this scheduling, once the resources (CPU cycles) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state. In case of non-preemptive scheduling does not interrupt a process running CPU in middle of the execution. Instead, it waits till the process complete its CPU burst time and then it can allocate the CPU to another process.

Algorithms based on non-preemptive scheduling are: [Shortest Job First (SJF basically non preemptive)](https://www.geeksforgeeks.org/program-shortest-job-first-sjf-scheduling-set-1-non-preemptive/) and [Priority (non preemptive version)](https://www.geeksforgeeks.org/operating-system-priority-scheduling-different-arrival-time-set-2/), etc.



TYPES OF SCHEDULING :

# First Come First Serve Scheduling

In the "First come first serve" scheduling algorithm, as the name suggests, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.

* First Come First Serve, is just like **FIFO**(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
* This is used in [Batch Systems](https://www.studytonight.com/operating-system/types-of-os).
* It's **easy to understand and implement** programmatically, using a Queue data structure, where a new process enters through the **tail** of the queue, and the scheduler selects process from the **head** of the queue.
* A perfect real life example of FCFS scheduling is **buying tickets at ticket counter**.

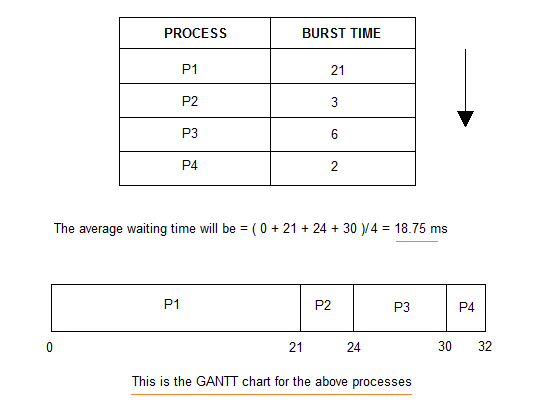
## Calculating Average Waiting Time

For every scheduling algorithm, **Average waiting time** is a crucial parameter to judge it's performance.

AWT or Average waiting time is the average of the waiting times of the processes in the queue, waiting for the scheduler to pick them for execution.

*Lower the Average Waiting Time, better the scheduling algorithm.*

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.



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.

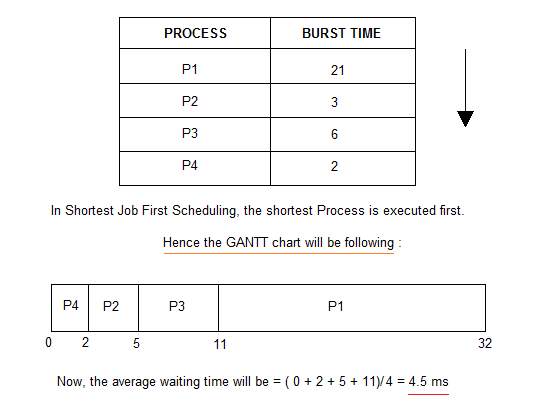
# Shortest Job First(SJF) Scheduling

Shortest Job First scheduling works on the process with the shortest **burst time** or **duration** first.

* This is the best approach to minimize waiting time.
* This is used in [Batch Systems](https://www.studytonight.com/operating-system/types-of-os).
* It is of two types:
  1. Non Pre-emptive
  2. Pre-emptive
* To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
* This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)

## 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**.

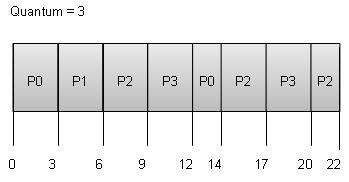


As you can see in the **GANTT chart** above, the process **P4** will be picked up first as it has the shortest burst time, then **P2**, followed by **P3** and at last **P1**.

We scheduled the same set of processes using the [First come first serve](https://www.studytonight.com/operating-system/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.

## Round Robin Scheduling

* 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 |

CODE:

FCFS.c :

#include<stdio.h>

void FCFS(int process[], int at[], int bt[], int n);

int main(){

    int n;

    printf("Enter no. of processes: ");

    scanf("%d",&n);

    int process[n],at[n],bt[n];

    int i;

    for(i=0;i<n;i++){

        process[i]=i;

        printf("Arrival time: ");scanf("%d",&at[i]);

        printf("Burst time: ");scanf("%d",&bt[i]);

    }

    FCFS(process,at,bt,n);

    return 1;

}

void FCFS(int process[], int at[], int bt[], int n){

    int wt[n],tat[n];

    int tmp;

    int i ,j;

    int total\_tat=0,total\_wt=0;

    float avg\_wt,avg\_tat;

    //Sorting according to arrival time

    for (i=0;i<n;i++){

        for(j=i;j<n;j++){

            if(at[i]>at[j])

            {

                tmp=at[i];at[i]=at[j];at[j]=tmp;

                tmp=bt[i];bt[i]=bt[j];bt[j]=tmp;

                tmp=process[i];process[i]=process[j];process[j]=tmp;

            }

        }

    }

    //Finding waiting time

    wt[0] = 0;  //For first arrived process

    for (int  i = 1; i < n ; i++ )

        wt[i] =  bt[i-1] + wt[i-1] ;

    //Finding turn around time

    for (int  i = 0; i < n ; i++)

        tat[i] = bt[i] + wt[i];

    //Finding avergage

    for (int  i=0; i<n; i++)

    {

        total\_wt = total\_wt + wt[i];

        total\_tat = total\_tat + tat[i];

    }

    avg\_tat = total\_tat/n;

    avg\_wt = total\_wt/n;

    //Showing in table

    printf("PID\tAT\tBT\tWT\tTAT\n");

    for (int  i=0; i<n; i++)

    {

        printf("%d\t%d\t%d\t%d\t%d\n",process[i],at[i],bt[i],wt[i],tat[i]);

    }

    printf("Average TAT: %f\n",avg\_tat);

    printf("Average WT: %f\n",avg\_wt);

}

SJF.c :

#include<stdio.h>

void SJF(int process[], int at[], int bt[], int n);

int main(){

    int n;

    printf("Enter no. of processes: ");

    scanf("%d",&n);

    int process[n],at[n],bt[n];

    int i;

    for(i=0;i<n;i++){

        process[i]=i;

        printf("Arrival time: ");scanf("%d",&at[i]);

        printf("Burst time: ");scanf("%d",&bt[i]);

    }

    SJF(process,at,bt,n);

    return 1;

}

void SJF(int process[], int at[], int bt[], int n){

    int wt[n],tat[n];

    int tmp;

    int i ,j;

    int total\_tat=0,total\_wt=0;

    float avg\_wt,avg\_tat;

    //Sorting according to arrival time

    for (i=0;i<n;i++){

        for(j=i;j<n;j++){

            if(at[i]>at[j])

            {

                tmp=at[i];at[i]=at[j];at[j]=tmp;

                tmp=bt[i];bt[i]=bt[j];bt[j]=tmp;

                tmp=process[i];process[i]=process[j];process[j]=tmp;

            }

            if(at[i]==at[j]){

                if(bt[i]>bt[j]){

                    tmp=at[i];at[i]=at[j];at[j]=tmp;

                    tmp=bt[i];bt[i]=bt[j];bt[j]=tmp;

                    tmp=process[i];process[i]=process[j];process[j]=tmp;

                }

            }

        }

    }

    //Finding waiting time

    wt[0] = 0;  //For first arrived process

    for (int  i = 1; i < n ; i++ )

        wt[i] =  bt[i-1] + wt[i-1] ;

    //Finding turn around time

    for (int  i = 0; i < n ; i++)

        tat[i] = bt[i] + wt[i];

    //Finding avergage

    for (int  i=0; i<n; i++)

    {

        total\_wt = total\_wt + wt[i];

        total\_tat = total\_tat + tat[i];

    }

    //Showing in table

    printf("PID\tAT\tBT\tWT\tTAT\n");

    for (int  i=0; i<n; i++)

    {

        printf("%d\t%d\t%d\t%d\t%d\n",process[i],at[i],bt[i],wt[i],tat[i]);

    }

    avg\_tat = total\_tat/n;

    avg\_wt = total\_wt/n;

    printf("Average TAT: %f\n",avg\_tat);

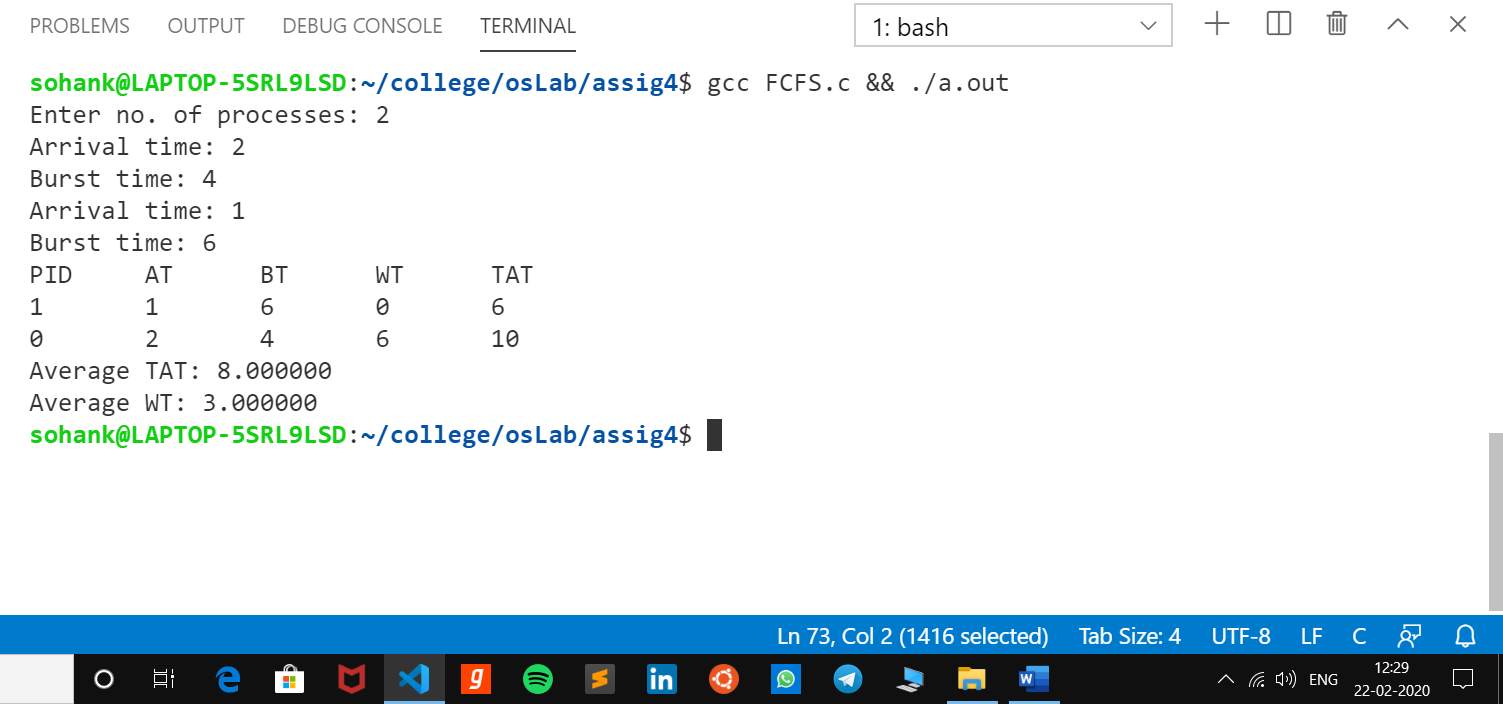
    printf("Average WT: %f\n",avg\_wt);

}

SJF.c :

OUTPUT :

FCFS:



SJF:

