	SHRI SANT GAJANAN MAHARAJ COLLEGE ENGG. SHEGAON		LABORATORY MANUAL
	PRACTICAL EXPERIMENT INSTRUCTION SHEET		
	EXPERIMENT TITLE: **FCFS CPU Scheduling Algorithm**		
EXPERIMENT NO.: SSGMCE/WI/ASH/01/1A6/01		ISSUE NO.:	**ISSUE DATE:**26.08.23
REV. DATE: 26.08.23	REV. NO.: 00	DEPTT. : INFORMATION TECHNOLOGY.	
LABORATORY: OPERATING SYSTEMS		SEMESTER : IV	PAGE NO. 01 OF 05

Date:

FCFS CPU SCHEDULING ALGORITHM

01. AIM :

Write a program to implement FCFS CPU scheduling algorithms.

02. FACILITIES :

Hardware: A Linux-based system (e.g., Ubuntu, Fedora, or any other distribution).

Software: C programming language, GCC compiler for compiling and running the program.

IDE/Editor: Any text editor or IDE that supports C programming (e.g., Visual Studio Code, Vim, Sublime Text, or Eclipse).

03. SCOPE :

This experiment focuses on implementing the **First-Come, First-Served (FCFS)** CPU scheduling algorithm. It includes:

1. Understanding how processes are handled in a **first-come** manner.
2. Calculating **Completion Time (CT)**, **Turnaround Time (TAT)**, and **Waiting Time (WT)** for each process.
3. Implementing the FCFS algorithm in C and demonstrating how it manages processes based on their arrival time.

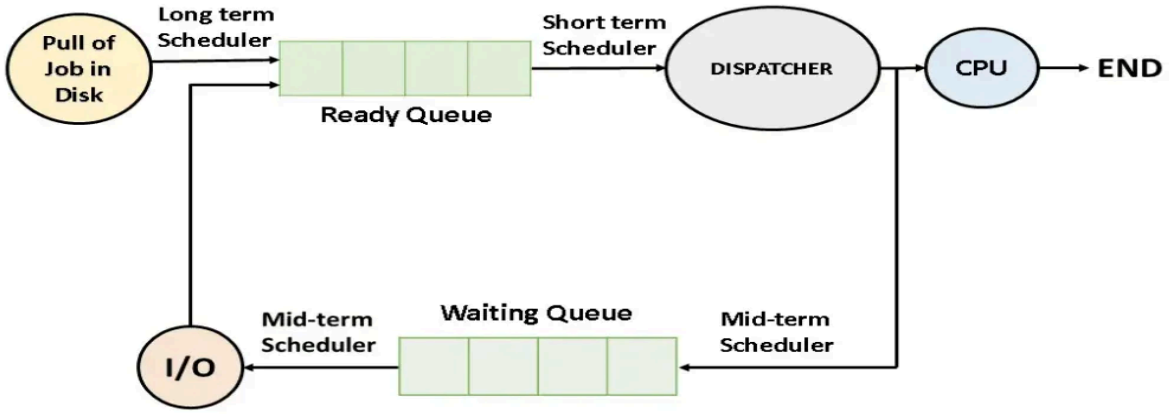
04 THEORY:

CPU Scheduling:

CPU Scheduling is the process of determining which process will execute next on the CPU. The efficiency of CPU scheduling directly impacts the overall performance of an operating system, as it influences metrics like throughput, response time, and fairness.

CPU Scheduling is important because a CPU can only handle one task at a time, but there are usually many tasks that need to be processed. The following are different purposes of a CPU scheduling time.

- Maximize the CPU utilization
- Minimize the response and waiting time of the process.



Key Terms in CPU Scheduling:

Different **CPU Scheduling algorithms** have different structures and the choice of a particular algorithm depends on a variety of factors.

1. **CPU Utilization:** The main purpose of any CPU algorithm is to keep the CPU as busy as possible. Theoretically, CPU usage can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the system load.
2. **Throughput:** The average CPU performance is the number of processes performed and completed during each unit. This is called throughput. The output may vary depending on the length or duration of the processes.
3. **Process:** A process is an instance of a program in execution, requiring resources such as CPU time, memory, and I/O devices. It moves through various states (like "ready," "running," and "terminated") during its lifecycle and performs tasks or computations.
4. **Ready Queue:** The ready queue is a list of processes that are in memory and waiting to be executed by the CPU. These processes are ready to run but are waiting for CPU resources to be allocated to them by the operating system's scheduler.
5. **Burst Time:** Burst time refers to the amount of CPU time a process needs to complete its execution. It is a key metric for scheduling algorithms, as the CPU allocates time based on the burst time required by processes.
6. **Turn Round Time:** For a particular process, the important conditions are how long it takes to perform that process. The time elapsed from the time of process delivery to the time of completion is known as the conversion time. Conversion time is the amount of time spent waiting for memory access, waiting in line, using CPU, and waiting for I / O.

```
Turnaround Time = Completion Time - Arrival Time
```

7. **Waiting Time:** The Scheduling algorithm does not affect the time required to complete the process once it has started performing. It only affects the waiting time of the process i.e. the time spent in the waiting process in the ready queue.

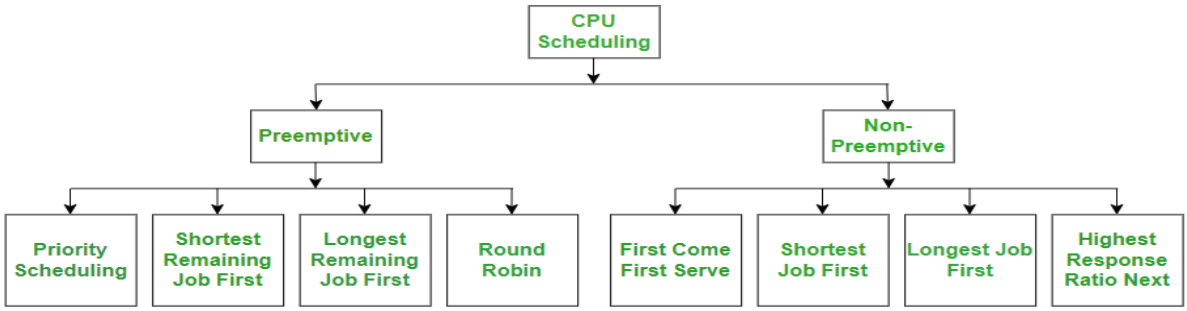
```
Waiting Time = Turnaround Time - Burst Time
```

8. **Response Time:** In a collaborative system, turn around time is not the best option. The process may produce something early and continue to compute the new results while the previous results are released to the user. Therefore another method is the time taken in the submission of the application process until the first response is issued. This measure is called response time.
9. **Context Switch:** A context switch is the process of saving the current process's state and loading the state of another process when the operating system switches between tasks. It allows multiple processes to share the CPU, but it introduces overhead due to the time spent on saving and restoring process states.

Different Types of CPU Scheduling Algorithms:

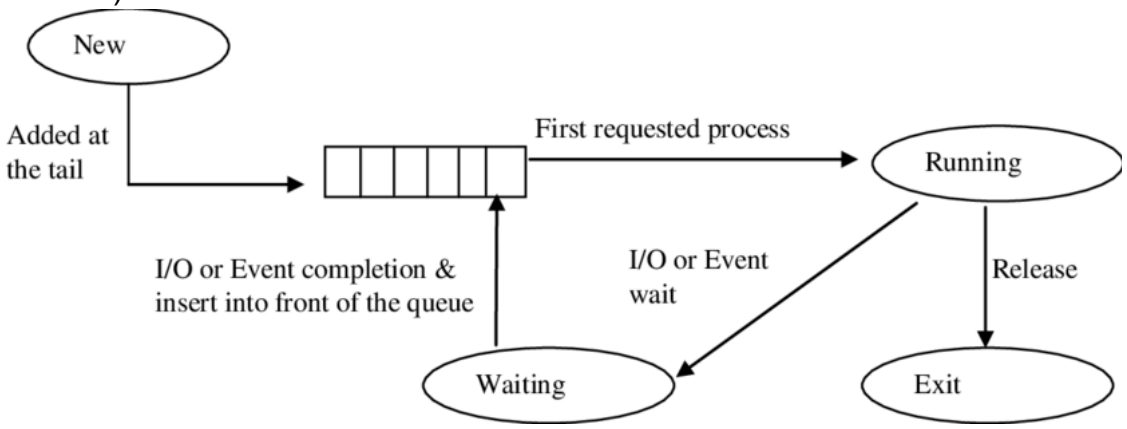
There are mainly two types of scheduling methods:

- **Preemptive Scheduling:** Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to the ready state.
- **Non-Preemptive Scheduling:** Non-Preemptive scheduling is used when a process terminates , or when a process switches from running state to waiting state.



First-Come, First-Served (FCFS):

FCFS is the simplest and one of the earliest CPU scheduling algorithms. In this approach, the process that arrives first gets executed first. This algorithm does not preemptively interrupt processes, meaning that once a process starts executing, it runs to completion unless it voluntarily relinquishes the CPU (e.g., by performing I/O operations).



Working of FCFS:

1. The processes are queued up in the order they arrive in the ready queue.
2. The CPU picks the first process in the ready queue and executes it until completion.
3. After the first process finishes execution, the next process in the queue is selected, and the cycle continues.
4. There is no preemption involved in FCFS; processes are executed in a non-interruptive manner.

Example:

Let's say you have 3 processes with the following details:

Process	Arrival Time (AT)	Burst Time (BT)
P1	0	5
P2	1	3
P3	2	2

Step 1: Sorting by Arrival Time (if not already sorted):

The processes are sorted in the order of arrival, so the order remains P1 → P2 → P3.

Step 2: Calculation:

P1: Completes at time 5 (Arrival time 0 + Burst time 5).
P2: Starts after P1 finishes at time 5, so it completes at time 8 (Start at 5 + Burst time 3).
P3: Starts after P2 finishes at time 8, so it completes at time 10 (Start at 8 + Burst time 2).

Step 3: Calculate Turnaround and Waiting Times:

TAT for P1: CT−AT=5−0=5CT - AT = 5 - 0 = 5CT−AT=5−0=5
TAT for P2: CT−AT=8−1=7CT - AT = 8 - 1 = 7CT−AT=8−1=7
TAT for P3: CT−AT=10−2=8CT - AT = 10 - 2 = 8CT−AT=10−2=8
WT for P1: TAT−BT=5−5=0TAT - BT = 5 - 5 = 0TAT−BT=5−5=0
WT for P2: TAT−BT=7−3=4TAT - BT = 7 - 3 = 4TAT−BT=7−3=4
WT for P3: TAT−BT=8−2=6TAT - BT = 8 - 2 = 6TAT−BT=8−2=6

Step 4: Output the Results:

Process	AT	BT	CT	TAT	WT
P1	0	5	5	5	0
P2	1	3	8	7	4
P3	2	2	10	8	6

05 PROGRAM:

```c
#include<stdio.h>

// Structure to hold Process Details
typedef struct {
    int pid;        // Process ID
    int arrival;    // Arrival Time
    int burst;      // Burst Time
    int completion; // Completion Time
    int turnaround; // Turnaround Time
    int waiting;    // Waiting Time
} Process;

// Function to calculate Completion Time, Turnaround Time, and Waiting Time
void findCompletionTime(Process p[], int n) {
    int time = 0;  // Tracks current time (CPU time)
    for (int i = 0; i < n; i++) {
        // If the current time is less than the arrival time, we move to the arrival time
        if (time < p[i].arrival) {
            time = p[i].arrival;
        }
        // Add the burst time of the process to the current time
        time += p[i].burst;
        p[i].completion = time;  // Set the completion time for the process
        p[i].turnaround = p[i].completion - p[i].arrival;  // Calculate Turnaround Time
        p[i].waiting = p[i].turnaround - p[i].burst;  // Calculate Waiting Time
    }
}

// Function to display Process Table
void displayProcessTable(Process p[], int n) {
    printf("\nProcess    AT      BT      CT      TAT      WT\n");
    for (int i = 0; i < n; i++) {
        // Displaying process information in a tabular format
        printf("P%d\t\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].completion, p[i].turnaround, p[i].waiting);
    }
}

int main() {
    int n;  // Number of processes
    printf("Enter number of processes: ");
    scanf("%d", &n);

    // Declaring array of processes
    Process p[n];

    // Input Arrival Time and Burst Time for each process
    printf("Enter Arrival Time and Burst Time for each process: \n");
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;  // Assign process ID starting from 1
        printf("P%d Arrival Time: ", i + 1);
        scanf("%d", &p[i].arrival);
        printf("P%d Burst Time: ", i + 1);
        scanf("%d", &p[i].burst);
    }
```

```
55
56      // Sorting processes based on their Arrival Time in ascending order
57      for (int i = 0; i < n - 1; i++) {
58          for (int j = 0; j < n - i - 1; j++) {
59              if (p[j].arrival > p[j + 1].arrival) {  // Swap if Arrival Time is
                    greater
60                  Process temp = p[j];
61                  p[j] = p[j + 1];
62                  p[j + 1] = temp;
63              }
64          }
65      }
66
67      // Calculate Completion Time, Turnaround Time, and Waiting Time for each
            process
68      findCompletionTime(p, n);
69
70      // Display the Process Table
71      displayProcessTable(p, n);
72
73      return 0;
74  }
```

06 OUTPUT

```
Enter number of processes: 3
Enter Arrival Time and Burst Time for each process:
P1 Arrival Time: 0
P1 Burst Time: 5
P2 Arrival Time: 1
P2 Burst Time: 3
P3 Arrival Time: 2
P3 Burst Time: 2

Process    AT      BT      CT      TAT      WT
P1          0       5       5       5       0
P2          1       3       8       7       4
P3          2       2       10      8       6
```

08 CONCLUSION:

This program simulates the **FCFS CPU Scheduling Algorithm** by calculating and displaying **Completion Time, Turnaround Time**, and **Waiting Time** for each process based on their arrival and burst times. FCFS is a simple, **non-preemptive** algorithm that's easy to implement, but it can lead to long waiting times for shorter processes if they arrive after longer ones.

09 VIVA QUESTIONS:

PREPARED BY **H. P. Amle**	**APPROVED BY** **HOD**