EXPERIMENT NO: 3 **DATE –**

Non-Pre-emptive CPU Scheduling Algorithm

(a) First Come First Serve CPU Scheduling Algorithm

Aim-To implement First come First serve CPU scheduling algorithm

THEORY –

Non-Preemptive CPU Scheduling

Non-preemptive scheduling is a CPU scheduling technique in which once a process starts execution on the CPU, it cannot be interrupted until it either completes execution or voluntarily gives up the CPU (for example, by performing an I/O operation). The CPU is not forcibly taken away from a running process.

- **Key Characteristics**:
    1. Simple to implement and has very low scheduling overhead.
    2. Provides predictability, as processes run in the order they are scheduled without being preempted.
    3. Can lead to the **convoy effect**, where small processes wait for a long process to complete, reducing overall responsiveness.
    4. Mostly used in **batch processing systems**, where execution order is more important than fast response time.

First-Come, First-Serve (FCFS) Scheduling

FCFS is the **simplest type of non-preemptive scheduling**. Processes are queued in the order they arrive in the ready queue, and the CPU is allocated to the process that has been waiting the longest (FIFO principle).

- **How it Works**:
    1. The ready queue is maintained as a **FIFO queue**.
    2. The process that arrives first gets executed first.

3. No preemption occurs—once a process starts, it will run until completion.

- **Performance Metrics**:
  - **Completion Time (CT):** The time at which a process finishes execution.
  - **Turnaround Time (TAT):** The total time spent in the system (from arrival to completion).

TAT=CT−ATAT

  - **Waiting Time (WT):** The time a process spends waiting in the ready queue.

WT=TAT−BT

EXAMPLE –

Process	**BT**
P_1	10
P_2	6
P_3	2
P_h	8

-GANTT diagram

P1	P2	P3	P_h

0 10 16 18 26

ATAT =0

PROCESS	BURST TIME	TURN AROUND TIME (CT-ATAT)	WAIT TIME (TAT-BT)
P1	10	10	0
P2	6	16	10
P3	2	18	16
P4	8	26	18

AVERAGE TURN AROUND TIME = (10+16+18+26)/4 = 17.5

AVERAGE WAIT TIME = (0+10+16+18)/4 = 11

PROGRAM –

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
using namespace std;

struct Process {
    string name;
    int arrival, burst, completion, turnaround, waiting;
};

int main() {
    int n;
    cout << "Enter number of processes: ";
    cin >> n;

    vector<Process> p(n);
    for (int i = 0; i < n; i++) {
        cout << "\nEnter Process Name, Arrival Time, Burst Time for P" << i + 1 << ": ";
        cin >> p[i].name >> p[i].arrival >> p[i].burst;
    }

    sort(p.begin(), p.end(), [](Process &a, Process &b) {
        return a.arrival < b.arrival;
    });

    int time = 0;
    float avgTAT = 0, avgWT = 0;
    int totalTAT = 0;
    vector<string> ganttOrder;
    vector<int> ganttTime;

    for (int i = 0; i < n; i++) {
        if (time < p[i].arrival) {
            ganttOrder.push_back("Idle");
            ganttTime.push_back(p[i].arrival);
            time = p[i].arrival;
        }
        time += p[i].burst;
        p[i].completion = time;
        p[i].turnaround = p[i].completion - p[i].arrival;
        p[i].waiting = p[i].turnaround - p[i].burst;
        avgTAT += p[i].turnaround;
        avgWT += p[i].waiting;
        totalTAT += p[i].turnaround;
        ganttOrder.push_back(p[i].name);
        ganttTime.push_back(time);
    }

    cout << "\n-------------------------------------------------------------\n";
    cout << "Process\tAT\tBT\tCT\tTAT\tWT\n";
    cout << "-------------------------------------------------------------\n";
    for (int i = 0; i < n; i++) {
        cout << p[i].name << "\t" << p[i].arrival << "\t" << p[i].burst << "\t"
             << p[i].completion << "\t" << p[i].turnaround << "\t" << p[i].waiting << "\n";
    }
    cout << "-------------------------------------------------------------\n";
    cout << "Total Turnaround Time = " << totalTAT << endl;
    cout << "Average Turnaround Time = " << avgTAT / n << endl;
    cout << "Average Waiting Time = " << avgWT / n << endl;
```

```
    cout << "\nGantt Chart:\n";
    cout << "--------------------------------------------------------------\n";
    for (auto &proc : ganttOrder) {
        cout << "| " << setw(5) << proc << " ";
    }
    cout << "|\n";
    cout << "--------------------------------------------------------------\n";
    cout << 0;
    for (auto &t : ganttTime) {
        cout << setw(7) << t;
    }
    cout << "\n";

    return 0;
}
```

OUTPUT –

```
PS C:\Users\Athar\OneDrive\Documents\college\SEM5\OS\LAB\exp3> cd "
Enter number of processes: 4

Enter Process Name, Arrival Time, Burst Time for P1: A
0
8

Enter Process Name, Arrival Time, Burst Time for P2: B
1
4

Enter Process Name, Arrival Time, Burst Time for P3: C
2
9

Enter Process Name, Arrival Time, Burst Time for P4: D
3
5

----------------------------------------------------------------
Process AT      BT      CT      TAT     WT
----------------------------------------------------------------
A       0       8       8       8       0
B       1       4       12      11      7
C       2       9       21      19      10
D       3       5       26      23      18
----------------------------------------------------------------
Total Turnaround Time = 61
Average Turnaround Time = 15.25
Average Waiting Time = 8.75

Gantt Chart:
----------------------------------------------------------------
|     A |     B |     C |     D |
----------------------------------------------------------------
0      8      12      21      26
```

CONCLUSION - This experiment successfully demonstrates the fundamental behavior of the FCFS algorithm: while it is simple and fair, its non-preemptive nature leads to high average waiting times and the convoy effect, making it inefficient for interactive systems.