**EXPERIMENT NO:** 2 **DATE:**

**TITLE: Non–Pre-emptive CPU Scheduling Algorithm**

**AIM:** To implement **First Come First Serve (FCFS)** Scheduling Algorithm.

**THEORY**

**What is Non-Pre-emptive CPU Scheduling Algorithm?**

A **Non-Pre-emptive CPU Scheduling Algorithm** is a scheduling method in which once a process starts execution on the CPU, it **cannot be interrupted** until it finishes its execution. The CPU will not be given to another process, even if another process arrives with higher priority.

This is in contrast to **pre-emptive scheduling**, where a running process can be interrupted and moved back to the ready queue if a higher-priority process arrives.

**First Come First Serve (FCFS) Scheduling**

**Working Principle**

- FCFS is the **simplest non-pre-emptive scheduling algorithm**.
- It follows the **FIFO (First In, First Out)** rule:
    1. Processes are executed in the order they arrive in the ready queue.
    2. The process that arrives first gets the CPU first.
    3. Once a process starts execution, it runs till completion before the next process starts.
- It is fair, but may lead to the **Convoy Effect** (long processes delaying shorter ones).

**Steps for FCFS Scheduling**

1. **Sort** processes according to **Arrival Time (AT)**.
2. **Calculate Completion Time (CT)**: The time at which the process finishes execution.
3. **Calculate Turnaround Time (TAT)**:

TAT=CT−AT

**Calculate Waiting Time (WT)**:

WT=TAT−BT

**Calculate Averages** for WT and TAT.

**Solved Example**

**Given:**

| Process | Burst Time (BT) | Arrival Time (AT) |
|---|---|---|
| P1 | 5 | 0 |
| P2 | 3 | 1 |
| P3 | 8 | 2 |

**Step 1: Sort by Arrival Time (Already Sorted)**

**Step 2: Calculate Completion Time (CT)**

- **P1**: CT = 0 + 5 = 5
- **P2**: CT = 5 + 3 = 8
- **P3**: CT = 8 + 8 = 16

**Step 3: Calculate TAT**

TAT=CT−AT

- P1: 5 − 0 = **5**
- P2: 8 − 1 = **7**
- P3: 16 − 2 = **14**

**Step 4: Calculate WT**

WT=TAT−BT

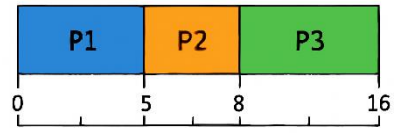
- P1: 5 − 5 = **0**
- P2: 7 − 3 = **4**
- P3: 14 − 8 = **6**

**Step 5: Calculate Averages**

- **Average WT (AWT)** = (0 + 4 + 6) ÷ 3 = **3.33**
- **Average TAT (ATAT)** = (5 + 7 + 14) ÷ 3 = **8.67**

---

**Gantt Chart**




---

**Final Table**

| Process | BT | AT | CT | TAT | WT |
|---|---|---|---|---|---|
| P1 | 5 | 0 | 5 | 5 | 0 |
| P2 | 3 | 1 | 8 | 7 | 4 |
| P3 | 8 | 2 | 16 | 14 | 6 |

**Average Waiting Time (AWT)** = 3.33
**Average Turnaround Time (ATAT)** = 8.67

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

int order[10];

int average(int *matrix, int n) {
int avg = 0;
   for (int i = 0; i < n; i++)
avg += matrix[i];
   return avg / n;
}

void sort_arr(int arr[], int n) {
int copy[n];
   for (int i = 0; i < n; i++) {
order[i] = i;
      copy[i] = arr[i];
   }
   for (int i = 0; i < n - 1; i++) {
for (int j = i + 1; j < n; j++) {
if (copy[i] > copy[j]) {
int temp = copy[i];
copy[i] = copy[j];
            copy[j] = temp;

            int t = order[i];
order[i] = order[j];
order[j] = t;
         }
      }
   }
}

void calc(int burst[], int arr[], int n) {
sort_arr(arr, n);
   int complete[n], TAT[n], WT[n], time = 0;

   for (int i = 0; i < n; i++) {
time += burst[order[i]];
      complete[order[i]] = time; }
for (int i = 0; i < n; i++)
      TAT[i] = complete[i] - arr[i];
for (int i = 0; i < n; i++)
WT[i] = TAT[i] - burst[i];
cout << "\nGantt Chart:\n";
   cout << " ";
   for (int i = 0; i < n; i++)
cout << "-------";    cout <<
"\n|";
   for (int i = 0; i < n; i++)
      cout << "  P" << order[i] + 1 << "  |";
cout << "\n ";
   for (int i = 0; i < n; i++)
cout << "-------";    cout
<< "\n0";    for (int i = 0;
i < n; i++)
      cout << setw(7) << complete[order[i]];
cout << "\n\n";
   cout << "Process\tBT\tAT\tTAT\tWT\n";
   for (int i = 0; i < n; i++) {
cout << "P" << i + 1 << "\t"
         << burst[i] << "\t"
         << arr[i] << "\t"
         << TAT[i] << "\t"
         << WT[i] << endl; }
   cout << "\nAverage waiting time : " <<
average(WT, n);
   cout << "\nAverage turn-around time : " <<
average(TAT, n);
}
int main() {
   int n;
   cout << "Enter number of processes: ";    cin
>> n;

   int burst[n], arr[n];
   for (int i = 0; i < n; i++) {
      cout << "Process " << i + 1 << endl;
cout << "Enter Burst Time: ";
```

```
      cin >> burst[i];
      cout << "Enter Arrival Time: ";       cin >>
arr[i];
   }

   cout << endl;     calc(burst, arr,
n);     return 0;
}
```

**OUPUT:**

```
PS C:\Users\audum\Desktop\Operating Syst
t_Serve_Scheduling_Algorithm.cpp -o Firs
Enter number of processes: 4
Process 1
Enter Burst Time: 3
Enter Arrival Time: 1
Process 2
Enter Burst Time: 2
Enter Arrival Time: 3
Process 3
Enter Burst Time: 5
Enter Arrival Time: 0
Process 4
Enter Burst Time: 4
Enter Arrival Time: 4


Gantt Chart:
 ---------------------------
|  P3  |  P1  |  P2  |  P4  |
 ---------------------------
0      5      8     10     14

Process BT      AT      TAT     WT
P1      3       1       7       4
P2      2       3       7       5
P3      5       0       5       0
P4      4       4       10      6

Average waiting time : 3
Average turn-around time : 7
```

**Conclusion:**first come first serve cpu scheduling algorithm was implemented successfully in cpp.