

# 4ITRC2 Operating System Lab

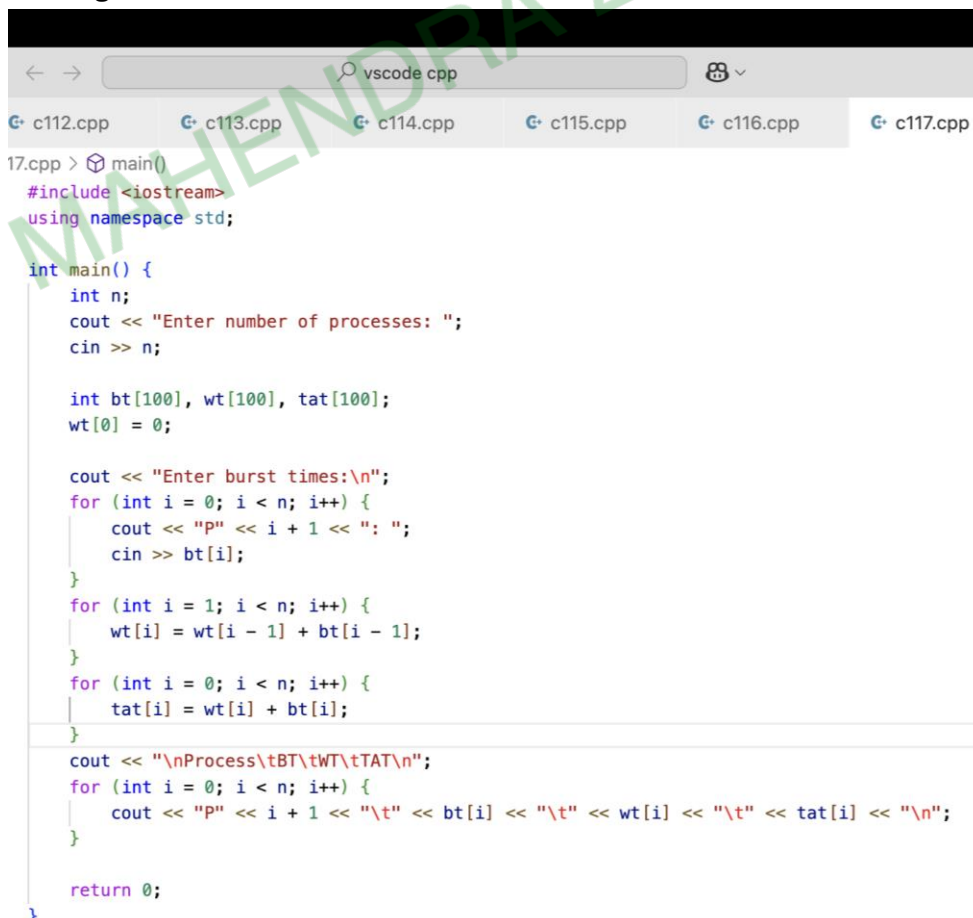
## Lab Assignment 5

Create C programs for the different scheduling algorithms.

- **Scheduling : CPU Scheduling** is the process of deciding **which process will use the CPU next** when multiple processes are ready to execute. It is a key function of the operating system that helps in **allocating CPU time efficiently** among all running processes.

It has many types but we are going to see only three in this document.

1. **First Come First Serve (FCFS) : First Come First Serve (FCFS)** is the simplest type of CPU scheduling algorithm. It executes processes **in the exact order** in which they arrive in the ready queue, similar to a queue at a ticket counter. The process that arrives first gets the CPU first. It is **non-preemptive**, meaning once a process starts executing, it runs till completion without interruption. FCFS is easy to implement but can lead to issues like **convoy effect**, where short jobs wait behind longer ones, increasing average waiting time.



```
17.cpp > main()
#include <iostream>
using namespace std;

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

    int bt[100], wt[100], tat[100];
    wt[0] = 0;

    cout << "Enter burst times:\n";
    for (int i = 0; i < n; i++) {
        cout << "P" << i + 1 << ": ";
        cin >> bt[i];
    }
    for (int i = 1; i < n; i++) {
        wt[i] = wt[i - 1] + bt[i - 1];
    }
    for (int i = 0; i < n; i++) {
        tat[i] = wt[i] + bt[i];
    }

    cout << "\nProcess\tBT\tWT\tTAT\n";
    for (int i = 0; i < n; i++) {
        cout << "P" << i + 1 << "\t" << bt[i] << "\t" << wt[i] << "\t" << tat[i] << "\n";
    }

    return 0;
}
```

```

psingh/Desktop/vscode/vscode cpp/"c117
Enter number of processes: 3
Enter burst times:
P1: 14
P2: 22
P3: 4

```

Process	BT	WT	TAT
P1	14	0	14
P2	22	14	36
P3	4	36	40

```

mahendrapratapsingh@mahendras-MacBook-Air vscode c

```

Ln 22, Col 6

2. **Shortest Job First (SJF) : Shortest Job First (SJF)** scheduling selects the process with the **smallest burst time** (execution time) from the ready queue. It is **non-preemptive**, meaning once the CPU is assigned, the process runs to completion. SJF is optimal in terms of **minimizing average waiting time**, but it requires **prior knowledge** of how long a process will take. If burst time predictions are accurate, this algorithm is very efficient. However, it can cause **starvation** for longer jobs if shorter ones keep arriving.

```

c118.cpp > ...
1  #include <iostream>
2  using namespace std;
3
4  void sortByBurstTime(int bt[], int p[], int n) {
5      for (int i = 0; i < n - 1; i++) {
6          for (int j = i + 1; j < n; j++) {
7              if (bt[i] > bt[j]) {
8                  swap(bt[i], bt[j]);
9                  swap(p[i], p[j]);
10             }
11         }
12     }
13 }
14
15 int main() {
16     int n;
17     cout << "Enter number of processes: ";
18     cin >> n;
19
20     int bt[100], wt[100], tat[100], p[100];
21
22     for (int i = 0; i < n; i++) {
23         p[i] = i + 1;
24         cout << "Enter burst time for P" << p[i] << ": ";
25         cin >> bt[i];
26     }
27
28     sortByBurstTime(bt, p, n);
29     wt[0] = 0;
30
31     for (int i = 1; i < n; i++) {

```

```

sortByBurstTime(bt, p, n);
wt[0] = 0;

for (int i = 1; i < n; i++) {
    wt[i] = wt[i - 1] + bt[i - 1];
}

for (int i = 0; i < n; i++) {
    tat[i] = wt[i] + bt[i];
}

cout << "\n--- Shortest Job First (SJF) Scheduling ---\n";
cout << "Process\tBT\tWT\tTAT\n";
for (int i = 0; i < n; i++) {
    cout << "P" << p[i] << "\t" << bt[i] << "\t" << wt[i] << "\t" << tat[i] << "\n";
}

return 0;
}

```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```

psingh/Desktop/vscode/vscode cpp/"c118
Enter number of processes: 3
Enter burst time for P1: 14
Enter burst time for P2: 22
Enter burst time for P3: 4

--- Shortest Job First (SJF) Scheduling ---
Process BT      WT      TAT
P3       4       0       4
P1       14      4       18
P2       22     18      40

```

mahendrapratapsingh@mahendras-MacBook-Air vscode cpp %

Ln 47, Col 1

- Round Robin Scheduling** : **Round Robin Scheduling** is a **preemptive** scheduling algorithm designed for **time-sharing systems**. Each process is assigned a fixed **time quantum** (e.g., 4ms), and the CPU cycles through the processes in the ready queue, giving each one a turn. If a process doesn't finish in its time slice, it is moved to the end of the queue. This ensures **fairness** and prevents any process from monopolizing the CPU. However, if the time quantum is too small, it can lead to **high context switching overhead**.

```

c114.cpp c115.cpp c116.cpp c117.cpp c11
:119.cpp > main()
#include <iostream>
using namespace std;

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

    int bt[100], rem_bt[100], wt[100] = {0}, tat[100] = {0}, p[100];

    for (int i = 0; i < n; i++) {
        p[i] = i + 1;
        cout << "Enter burst time for P" << p[i] << ": ";
        cin >> bt[i];
        rem_bt[i] = bt[i];
    }

    cout << "Enter time quantum: ";
    cin >> tq;

    int t = 0;
    bool done;

    do {
        done = true;
        for (int i = 0; i < n; i++) {
            if (rem_bt[i] > 0) {

```

```

24         do {
25             done = true;
26             for (int i = 0; i < n; i++) {
27                 if (rem_bt[i] > 0) {
28                     done = false;
29                     if (rem_bt[i] > tq) {
30                         t += tq;
31                         rem_bt[i] -= tq;
32                     } else {
33                         t += rem_bt[i];
34                         wt[i] = t - bt[i];
35                         rem_bt[i] = 0;
36                     }
37                 }
38             }
39         } while (!done);
40
41         for (int i = 0; i < n; i++) {
42             tat[i] = bt[i] + wt[i];
43         }
44
45         cout << "\n--- Round Robin Scheduling ---\n";
46         cout << "Process\tBT\tWT\tTAT\n";
47         for (int i = 0; i < n; i++) {
48             cout << "P" << p[i] << "\t" << bt[i] << "\t" << wt[i] << "\t" << tat[i] << "\n";
49         }
50
51         return 0;
52     }

```

PROBLEMS    OUTPUT    DEBUG CONSOLE

```
Enter number of processes: 3
Enter burst time for P1: 14
Enter burst time for P2: 22
Enter burst time for P3: 4
Enter time quantum: 5
```

--- Round Robin Scheduling ---

Process	BT	WT	TAT
P1	14	14	28
P2	22	18	40
P3	4	10	14

○ mahendrapratapsingh@mahendras-MacBoo



MAHENDRA 2314039