***Name : Sohag Ali***

***ID : 0242220005101426***

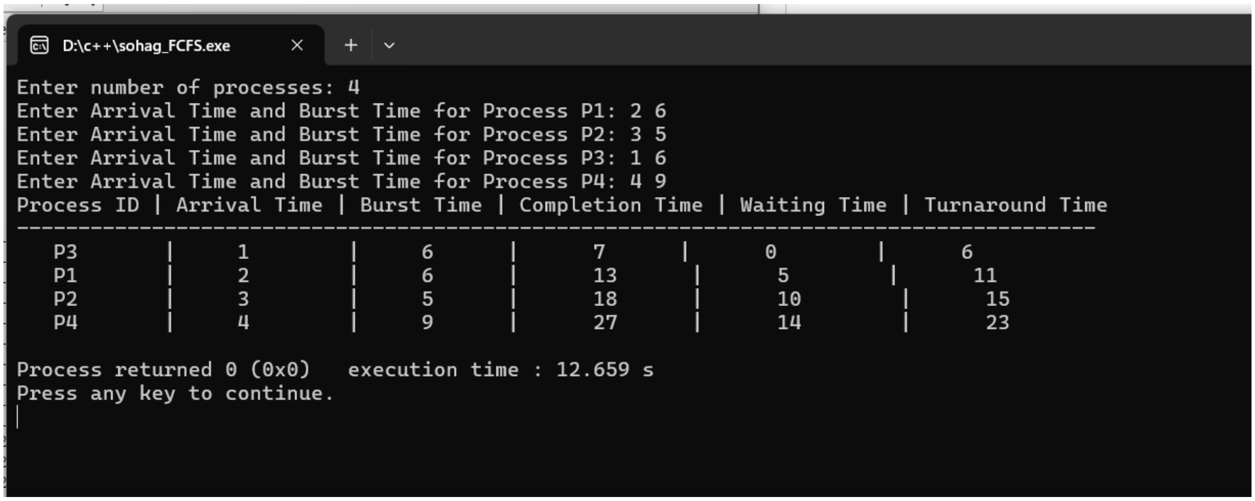
***Section : 63_B***

## Q1. FCFS with idle time and arrival time

```
#include <iostream>
#include <algorithm>
using namespace std;
struct Process {
int id;
int arrivalTime;
int burstTime;
int completionTime;
int waitingTime;
int turnAroundTime;
};
void findCompletionTime(Process processes[], int n) {
sort(processes, processes + n, [](Process a, Process b) {
return a.arrivalTime < b.arrivalTime;
});
int currentTime = 0;
for (int i = 0; i < n; i++) {
if (currentTime < processes[i].arrivalTime) {
       currentTime = processes[i].arrivalTime;
     }
```

```
        processes[i].completionTime = currentTime + processes[i].burstTime;
        currentTime = processes[i].completionTime;

        processes[i].turnAroundTime = processes[i].completionTime - processes[i].arrivalTime;
        processes[i].waitingTime = processes[i].turnAroundTime - processes[i].burstTime;
    }
}

void displayResults(Process processes[], int n) {
    cout << "Process ID | Arrival Time | Burst Time | Completion Time | Waiting Time | Turnaround Time\n";
    cout << "----------------------------------------------------------------------------------------\n";
    for (int i = 0; i < n; i++) {
        cout << "   P" << processes[i].id << "       |     "
             << processes[i].arrivalTime << "        |     "
             << processes[i].burstTime << "      |      "
             << processes[i].completionTime << "      |      "
             << processes[i].waitingTime << "        |      "
             << processes[i].turnAroundTime << "\n";
    }
}

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

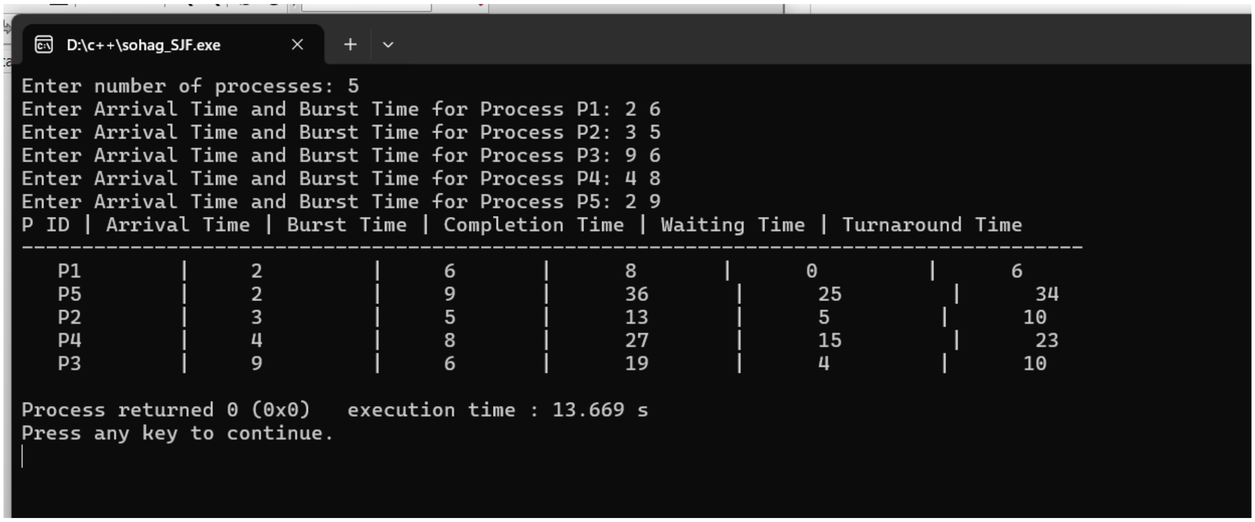
```
Process processes[n];

for (int i = 0; i < n; i++) {

processes[i].id = i + 1;

cout << "Enter Arrival Time and Burst Time for Process P" << i + 1 << ": ";

cin >> processes[i].arrivalTime >> processes[i].burstTime;

}

findCompletionTime(processes, n);

displayResults(processes, n);

return 0;

}
```



```
Enter number of processes: 4
Enter Arrival Time and Burst Time for Process P1: 2 6
Enter Arrival Time and Burst Time for Process P2: 3 5
Enter Arrival Time and Burst Time for Process P3: 1 6
Enter Arrival Time and Burst Time for Process P4: 4 9
Process ID | Arrival Time | Burst Time | Completion Time | Waiting Time | Turnaround Time
-------------------------------------------------------------------------------------
   P3      |     1         |     6      |      7      |      0        |      6
   P1      |     2         |     6      |      13      |      5        |      11
   P2      |     3         |     5      |      18      |      10        |      15
   P4      |     4         |     9      |      27      |      14        |      23

Process returned 0 (0x0)   execution time : 12.659 s
Press any key to continue.
```

## Q2. SJF with arrival time and handle idle issue

```
#include <iostream>

#include <climits>

#include <algorithm>


using namespace std;


struct Process {
    int id;
    int arrivalTime;
    int burstTime;
    int completionTime;
    int waitingTime;
    int turnAroundTime;
    bool isCompleted;
};

bool compareArrival(Process a, Process b) {
    return a.arrivalTime < b.arrivalTime;
}

void findCompletionTime(Process processes[], int n) {
    int currentTime = 0;
    int completed = 0;

    while (completed < n) {
        int idx = -1;
        int minBurstTime = INT_MAX;

        for (int i = 0; i < n; i++) {
            if (!processes[i].isCompleted && processes[i].arrivalTime <= currentTime) {
                if (processes[i].burstTime < minBurstTime) {
                    minBurstTime = processes[i].burstTime;
                    idx = i;
                }
            }
        }


        if (idx != -1) {
            processes[idx].isCompleted = true;
            processes[idx].completionTime = currentTime + processes[idx].burstTime;
            currentTime = processes[idx].completionTime;
            processes[idx].turnAroundTime = processes[idx].completionTime - processes[idx].arrivalTime;
            processes[idx].waitingTime = processes[idx].turnAroundTime - processes[idx].burstTime;
            completed++;
        } else {

            currentTime++;
        }
    }
}

void displayResults(Process processes[], int n) {
    cout << "P ID | Arrival Time | Burst Time | Completion Time | Waiting Time | Turnaround Time\n";
    cout << "-----------------------------------------------------------------------------------------\n";
    for (int i = 0; i < n; i++) {
        cout << "   P" << processes[i].id << "       |     "
             << processes[i].arrivalTime << "         |     "
        << processes[i].burstTime << "       |      "
        << processes[i].completionTime << "       |      "
        << processes[i].waitingTime << "         |      "
<< processes[i].turnAroundTime << "\n";
}
}
int main() {
int n;
cout << "Enter number of processes: ";
cin >> n;
Process processes[n];
for (int i = 0; i < n; i++) {
processes[i].id = i + 1;
processes[i].isCompleted = false;
cout << "Enter Arrival Time and Burst Time for Process P" << i + 1 << ": ";
cin >> processes[i].arrivalTime >> processes[i].burstTime;
}
sort(processes, processes + n, compareArrival);
findCompletionTime(processes, n);
displayResults(processes, n);
return 0;
}
```



```
Enter number of processes: 5
Enter Arrival Time and Burst Time for Process P1: 2 6
Enter Arrival Time and Burst Time for Process P2: 3 5
Enter Arrival Time and Burst Time for Process P3: 9 6
Enter Arrival Time and Burst Time for Process P4: 4 8
Enter Arrival Time and Burst Time for Process P5: 2 9
P ID | Arrival Time | Burst Time | Completion Time | Waiting Time | Turnaround Time
----------------------------------------------------------------------------------------
   P1        |     2         |     6       |      8       |      0         |      6
   P5        |     2         |     9       |      36       |       25        |       34
   P2        |     3         |     5       |      13       |      5         |       10
   P4        |     4         |     8       |      27       |       15        |       23
   P3        |     9         |     6       |      19       |      4         |       10

Process returned 0 (0x0)   execution time : 13.669 s
Press any key to continue.
```

## Q3. Priority Scheduling

```
#include <iostream>

#include <climits>

#include <algorithm>

using namespace std;

struct Process {

int id;

int arrivalTime;

int burstTime;

int completionTime;

int waitingTime;

int turnAroundTime;

bool isCompleted;

int priority;

};

bool compareArrival(Process a, Process b) {

return a.arrivalTime < b.arrivalTime;

}
```

```
void findCompletionTime(Process processes[], int n) {
    int currentTime = 0;
    int completed = 0;

    while (completed < n) {
        int idx = -1;
        int highpriority = INT_MAX;


        for (int i = 0; i < n; i++) {
            if (!processes[i].isCompleted && processes[i].arrivalTime <= currentTime) {
                if (processes[i].priority < highpriority) {
                    highpriority = processes[i].priority;
                    idx = i;
                }
            }
        }


        if (idx != -1) {
            processes[idx].isCompleted = true;
            processes[idx].completionTime = currentTime + processes[idx].burstTime;
            currentTime = processes[idx].completionTime;
            processes[idx].turnAroundTime = processes[idx].completionTime -
processes[idx].arrivalTime;
```

```
            processes[idx].waitingTime = processes[idx].turnAroundTime - processes[idx].burstTime;
            completed++;
        } else {

            currentTime++;
        }
    }
}

void displayResults(Process processes[], int n) {
    cout << "P ID | Arrival Time | Burst Time | Completion Time | Waiting Time | Turnaround Time\n";
    cout << "-------------------------------------------------------------------------------------\n";
    for (int i = 0; i < n; i++) {
        cout << "   P" << processes[i].id << "        |     "
             << processes[i].arrivalTime << "         |     "
             << processes[i].burstTime << "       |      "
             << processes[i].completionTime << "       |      "
             << processes[i].waitingTime << "         |      "
             << processes[i].turnAroundTime << "\n";
    }
}

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

    Process processes[n];

    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        processes[i].isCompleted = false;
        cout << "Enter Arrival Time and Burst Time & priority for Process P" << i + 1 << ": ";
cin >> processes[i].arrivalTime >> processes[i].burstTime>>processes[i].priority;
}
sort(processes, processes + n, compareArrival);
findCompletionTime(processes, n);
displayResults(processes, n);
return 0;
}
```