CPU Scheduling Algorithms

Objectives:

- **First-Come, First-Served (FCFS)** Scheduling: FCFS algorithm schedules processes based on their arrival times. It executes the processes in the order they arrive, without preemption.
- **Shortest Job First (SJF)** Scheduling: SJF algorithm schedules processes based on their burst times. The process with the shortest burst time is selected for execution first. This implementation is non-preemptive.

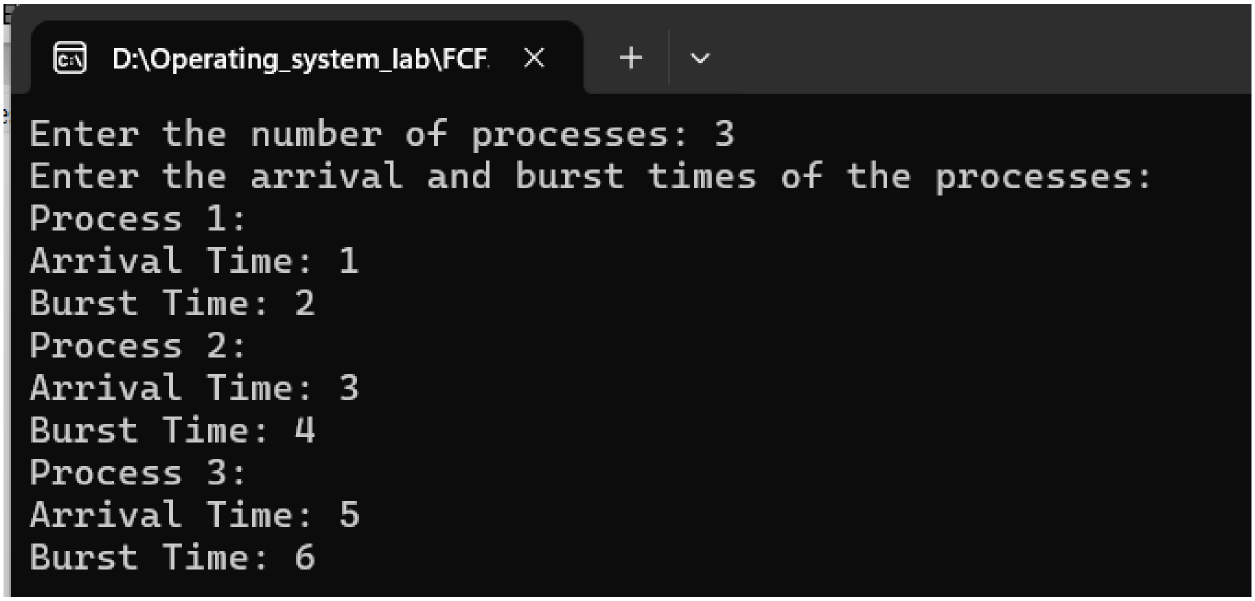
Both algorithms are essential in understanding process scheduling in operating systems and serve to demonstrate the impact of scheduling strategies on waiting times and turnaround times of processes.

01.First-Come, First-Served (FCFS) Scheduling.

Code:

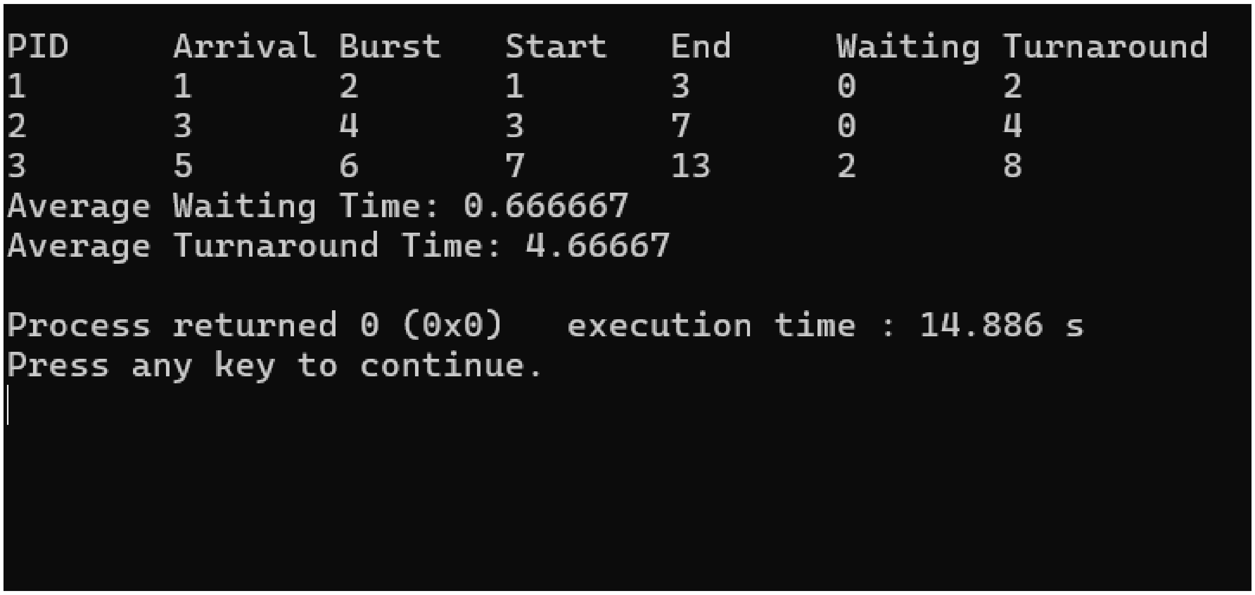
```
#include <iostream>
using namespace std;

void FCFS(int burstTimes[], int arrivalTimes[], int n) {
   int waitingTime[n], turnaroundTime[n], startTime[n], endTime[n];
   int totalWaitingTime = 0, totalTurnaroundTime = 0;

   // Calculate start time, end time, and waiting time
   startTime[0] = arrivalTimes[0];
   endTime[0] = startTime[0] + burstTimes[0];
   waitingTime[0] = 0;

   for (int i = 1; i < n; i++) {
       startTime[i] = max(endTime[i - 1], arrivalTimes[i]);
       endTime[i] = startTime[i] + burstTimes[i];
       waitingTime[i] = startTime[i] - arrivalTimes[i];
       if (waitingTime[i] < 0) waitingTime[i] = 0; // Ensure no negative waiting time
   }

   // Calculate turnaround time
   for (int i = 0; i < n; i++) {
       turnaroundTime[i] = burstTimes[i] + waitingTime[i];
       totalWaitingTime += waitingTime[i];
       totalTurnaroundTime += turnaroundTime[i];
   }

   // Output results
   cout << "\nPID\tArrival\tBurst\tStart\tEnd\tWaiting\tTurnaround\n";
   for (int i = 0; i < n; i++) {
       cout << i + 1 << "\t" << arrivalTimes[i] << "\t" << burstTimes[i] << "\t"
            << startTime[i] << "\t" << endTime[i] << "\t" << waitingTime[i] << "\t" << turnaroundTime[i] << "\n";
   }

   cout << "Average Waiting Time: " << (float)totalWaitingTime / n << "\n";
   cout << "Average Turnaround Time: " << (float)totalTurnaroundTime / n << "\n";
}

int main() {
   int n;
   cout << "Enter the number of processes: ";
   cin >> n;
   int burstTimes[n], arrivalTimes[n];
   cout << "Enter the arrival and burst times of the processes:\n";
   for (int i = 0; i < n; i++) {
       cout << "Process " << i + 1 << ":\n";
       cout << "Arrival Time: ";
       cin >> arrivalTimes[i];
       cout << "Burst Time: ";
       cin >> burstTimes[i];
   }

   FCFS(burstTimes, arrivalTimes, n);

   return 0;
}
```

Input:



```
Enter the number of processes: 3
Enter the arrival and burst times of the processes:
Process 1:
Arrival Time: 1
Burst Time: 2
Process 2:
Arrival Time: 3
Burst Time: 4
Process 3:
Arrival Time: 5
Burst Time: 6
```

Output:



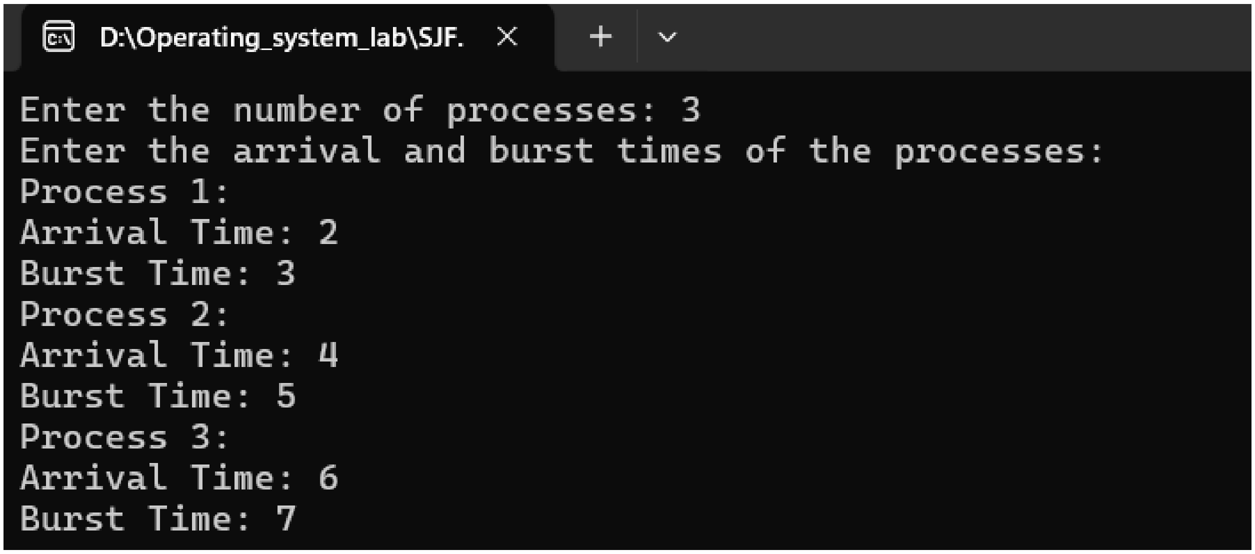
```
PID     Arrival Burst   Start   End     Waiting Turnaround
1       1       2       1       3       0       2
2       3       4       3       7       0       4
3       5       6       7       13      2       8
Average Waiting Time: 0.666667
Average Turnaround Time: 4.66667

Process returned 0 (0x0)   execution time : 14.886 s
Press any key to continue.
```

02.Shortest Job First (SJF) Scheduling.

Code:

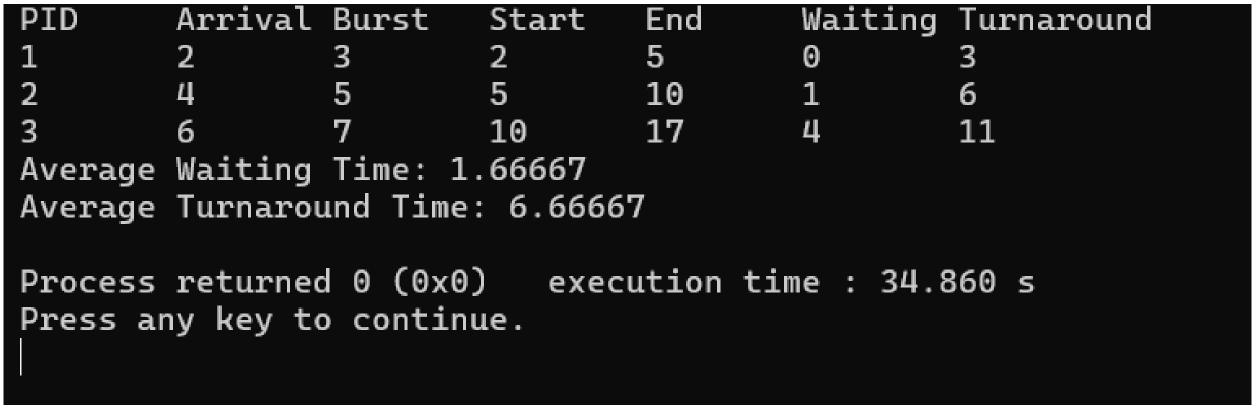
```
#include <iostream>
using namespace std;

void SJF(int burstTimes[], int arrivalTimes[], int n) {
    int waitingTime[n], turnaroundTime[n], startTime[n], endTime[n];
    int totalWaitingTime = 0, totalTurnaroundTime = 0;

    // Sort processes by burst time (non-preemptive)
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (burstTimes[i] > burstTimes[j]) {
                swap(burstTimes[i], burstTimes[j]);
                swap(arrivalTimes[i], arrivalTimes[j]);
            }
        }
    }
    // Calculate start time, end time, and waiting time
    startTime[0] = arrivalTimes[0];
    endTime[0] = startTime[0] + burstTimes[0];
    waitingTime[0] = 0;

    for (int i = 1; i < n; i++) {
        startTime[i] = max(endTime[i - 1], arrivalTimes[i]);
        endTime[i] = startTime[i] + burstTimes[i];
        waitingTime[i] = startTime[i] - arrivalTimes[i];
        if (waitingTime[i] < 0) waitingTime[i] = 0;
    }

    // Calculate turnaround time
    for (int i = 0; i < n; i++) {
        turnaroundTime[i] = burstTimes[i] + waitingTime[i];
        totalWaitingTime += waitingTime[i];
        totalTurnaroundTime += turnaroundTime[i];
    }

    // Output results
    cout << "\nPID\tArrival\tBurst\tStart\tEnd\tWaiting\tTurnaround\n";
    for (int i = 0; i < n; i++) {
        cout << i + 1 << "\t" << arrivalTimes[i] << "\t" << burstTimes[i] << "\t"
             << startTime[i] << "\t" << endTime[i] << "\t" << waitingTime[i] << "\t" << turnaroundTime[i] << "\n";
    }

    cout << "Average Waiting Time: " << (float)totalWaitingTime / n << "\n";
    cout << "Average Turnaround Time: " << (float)totalTurnaroundTime / n << "\n";
}

int main() {
    int n;

    cout << "Enter the number of processes: ";
    cin >> n;

    int burstTimes[n], arrivalTimes[n];
    cout << "Enter the arrival and burst times of the processes:\n";
    for (int i = 0; i < n; i++) {
        cout << "Process " << i + 1 << ":\n";
        cout << "Arrival Time: ";
        cin >> arrivalTimes[i];
        cout << "Burst Time: ";
        cin >> burstTimes[i];
    }

    SJF(burstTimes, arrivalTimes, n);

    return 0;
}
```

Input:



```
Enter the number of processes: 3
Enter the arrival and burst times of the processes:
Process 1:
Arrival Time: 2
Burst Time: 3
Process 2:
Arrival Time: 4
Burst Time: 5
Process 3:
Arrival Time: 6
Burst Time: 7
```

Output:



```
PID     Arrival Burst   Start   End     Waiting Turnaround
1       2       3       2       5       0       3
2       4       5       5       10      1       6
3       6       7       10      17      4       11
Average Waiting Time: 1.66667
Average Turnaround Time: 6.66667

Process returned 0 (0x0)   execution time : 34.860 s
Press any key to continue.
```