A Lab Report of

Applied Operating Systems

Lab 01:"To implement the FCFS CPU Process Scheduling Algorithm when

i)No arrival time is given ii) Arrival Time is given"

Submitted By

Aayush Parajuli

BESE 2021

Roll No.: 03

Submitted To

Er. Pratikshya Shrestha

Department of Research and Development

GANDAKI COLLEGE OF ENGINEERING AND SCIENCE

Lamachaur, Kaski, Nepal

(May 29, 2024)

OBJECTIVE: To implement the FCFS CPU Process Scheduling Algorithm when i)No arrival time is given ii) Arrival Time is given

THEORY:

First-Come, First-Served (FCFS) Scheduling Algorithm

The First-Come, First-Served (FCFS) scheduling algorithm is one of the simplest types of CPU scheduling algorithms. In FCFS, the process that arrives first is executed first. It operates in a non-preemptive manner, meaning once a process starts its execution, it runs until completion.

Explanation of Key Metrics and Calculation:

1. **Completion Time (CT)**: The time at which a process completes its execution. For processes arriving at different times, CT is calculated based on the current time plus the burst time of the process.
2. **Turnaround Time (TAT)**: The total time taken for a process to complete its execution. It is calculated as:
   $TAT = CT - AT$
   where AT is the arrival time of the process.
3. **Waiting Time (WT)**: The total time a process spends waiting in the ready queue. It is calculated as:
   $WT = TAT - BT$
   where BT is the burst time of the process.

Algorithms

Scenario i) All processes arrive at 0 unit of time and have no arrival time given

Algorithm:

1. **Input**: Burst times of all processes.
2. **Initialize**:
   - Set the waiting time of the first process to 0.

3. **Calculate Completion Time (CT)**:
    - For the first process, CT is equal to its burst time.
    - For subsequent processes, CT is the sum of the previous process's CT and the current process's burst time.
4. **Calculate Turnaround Time (TAT)**:
    - TAT for each process is equal to its CT (since arrival time is 0).
5. **Calculate Waiting Time (WT)**:
    - WT for each process is calculated as $WT = TAT - BT$.
6. **Output**: The CT, TAT, and WT for each process along with the average TAT and WT.

Scenario ii) Processes have their arrival time given

Algorithm:

1. **Input**: Arrival times and burst times of all processes.
2. **Sort processes** by their arrival times.
3. **Initialize**:
    - Set the current time to 0.
4. **Calculate Completion Time (CT)**:
    - For each process, if the current time is less than the arrival time, set the current time to the arrival time.
    - Update CT for each process as the sum of the current time and the process's burst time.
    - Update the current time to the process's CT.
5. **Calculate Turnaround Time (TAT)**:
    - $TAT = CT - AT$
6. **Calculate Waiting Time (WT)**:
    - $WT = TAT - BT$
7. **Output**: The CT, TAT, and WT for each process along with the average TAT and WT

Code Implementation:

Scenario i) All processes arrive at 0 unit of time and have no arrival time given

```
#include <iostream>
#include <vector>

using namespace std;

struct Process {
    int id;
    int burstTime;
    int completionTime;
    int waitingTime;
    int turnAroundTime;
};

void calculateTimes(vector<Process>& processes) {
    int n = processes.size();
    int totalWT = 0, totalTAT = 0;

    // Calculating completion time for each process
    processes[0].completionTime = processes[0].burstTime;

    for (int i = 1; i < n; i++) {
        processes[i].completionTime = processes[i-1].completionTime +
processes[i].burstTime;
    }

    // Calculating TAT and WT for each process
    for (int i = 0; i < n; i++) {
        processes[i].turnAroundTime = processes[i].completionTime;
        processes[i].waitingTime = processes[i].turnAroundTime -
processes[i].burstTime;
        totalWT += processes[i].waitingTime;
        totalTAT += processes[i].turnAroundTime;
    }
```

```
    cout << "Process\tBurst Time\tCompletion Time\tWaiting
Time\tTurnaround Time\n";
    for (const auto& process : processes) {
        cout << process.id << "\t" << process.burstTime << "\t\t" <<
process.completionTime << "\t\t" << process.waitingTime << "\t\t" <<
process.turnAroundTime << "\n";
    }

    cout << "\nAverage Waiting Time: " << (float)totalWT / n << "\n";
    cout << "Average Turnaround Time: " << (float)totalTAT / n << "\n";
}

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

    vector<Process> processes(n);

    for (int i = 0; i < n; i++) {
        processes[i].id = i+1;
        cout << "Enter burst time for process " << i+1 << ": ";
        cin >> processes[i].burstTime;
    }

    calculateTimes(processes);

    return 0;
}
```

Output:

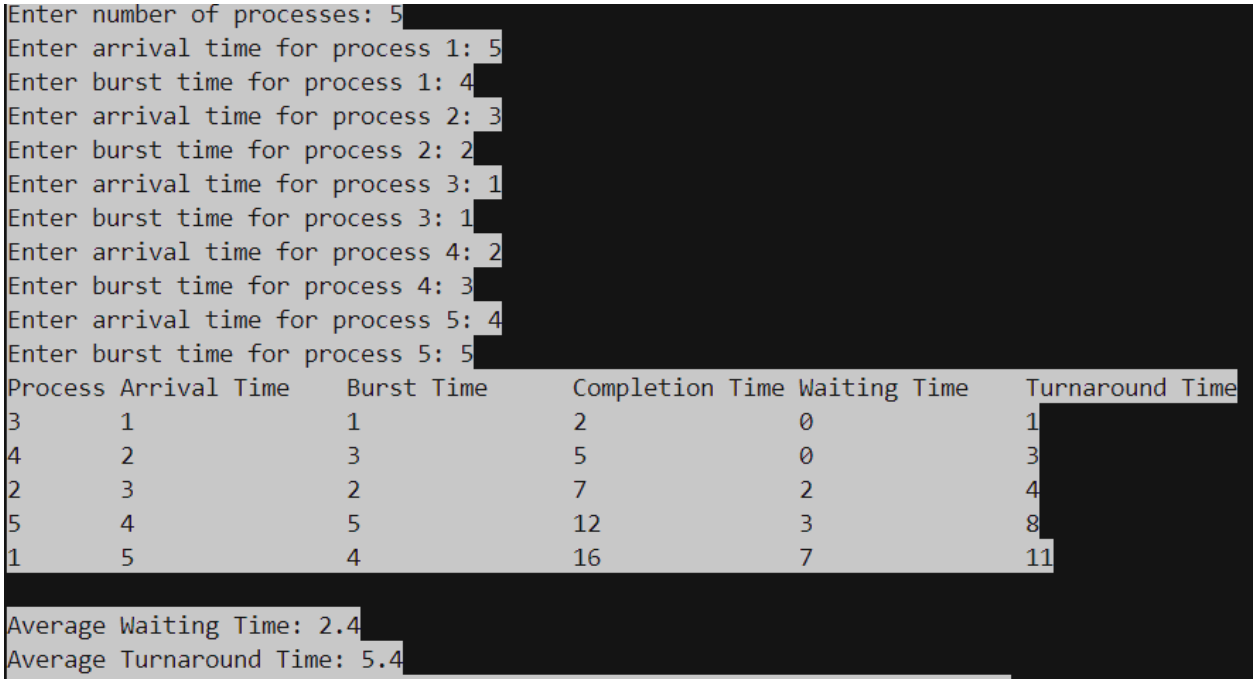
```
Enter number of processes: 5
Enter burst time for process 1: 5
Enter burst time for process 2: 4
Enter burst time for process 3: 3
Enter burst time for process 4: 2
Enter burst time for process 5: 1
Process Burst Time      Completion Time Waiting Time    Turnaround Time
1       5               5               0               5
2       4               9               5               9
3       3               12              9               12
4       2               14              12              14
5       1               15              14              15

Average Waiting Time: 8
Average Turnaround Time: 11
```

Scenario ii) Processes have their arrival time given

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

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

bool compareArrivalTime(const Process& p1, const Process& p2) {
    return p1.arrivalTime < p2.arrivalTime;
}

void calculateTimes(vector<Process>& processes) {
    int n = processes.size();
    int totalWT = 0, totalTAT = 0;
    int currentTime = 0;

    sort(processes.begin(), processes.end(), compareArrivalTime);

    for (int i = 0; i < n; i++) {
        if (currentTime < processes[i].arrivalTime) {
            currentTime = processes[i].arrivalTime;
        }
        processes[i].completionTime = currentTime +
processes[i].burstTime;
        processes[i].turnAroundTime = processes[i].completionTime -
processes[i].arrivalTime;
        processes[i].waitingTime = processes[i].turnAroundTime -
processes[i].burstTime;
        currentTime = processes[i].completionTime;
```

```
        totalWT += processes[i].waitingTime;
        totalTAT += processes[i].turnAroundTime;
    }

    cout << "Process\tArrival Time\tBurst Time\tCompletion Time\tWaiting
Time\tTurnaround Time\n";
    for (const auto& process : processes) {
        cout << process.id << "\t" << process.arrivalTime << "\t\t" <<
process.burstTime << "\t\t" << process.completionTime << "\t\t" <<
process.waitingTime << "\t\t" << process.turnAroundTime << "\n";
    }

    cout << "\nAverage Waiting Time: " << (float)totalWT / n << "\n";
    cout << "Average Turnaround Time: " << (float)totalTAT / n << "\n";
}

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

    vector<Process> processes(n);

    for (int i = 0; i < n; i++) {
        processes[i].id = i+1;
        cout << "Enter arrival time for process " << i+1 << ": ";
        cin >> processes[i].arrivalTime;
        cout << "Enter burst time for process " << i+1 << ": ";
        cin >> processes[i].burstTime;
    }

    calculateTimes(processes);

    return 0;
}
```

Output:



```
Enter number of processes: 5
Enter arrival time for process 1: 5
Enter burst time for process 1: 4
Enter arrival time for process 2: 3
Enter burst time for process 2: 2
Enter arrival time for process 3: 1
Enter burst time for process 3: 1
Enter arrival time for process 4: 2
Enter burst time for process 4: 3
Enter arrival time for process 5: 4
Enter burst time for process 5: 5
Process Arrival Time    Burst Time      Completion Time Waiting Time    Turnaround Time
3       1               1               2               0               1
4       2               3               5               0               3
2       3               2               7               2               4
5       4               5               12              3               8
1       5               4               16              7               11

Average Waiting Time: 2.4
Average Turnaround Time: 5.4
```

CONCLUSION:

In this lab, we explored the First-Come, First-Served (FCFS) scheduling algorithm, a fundamental and straightforward scheduling technique used in operating systems. Through the implementation and analysis of two different scenarios, we gained a deeper understanding of how FCFS operates and its impact on process scheduling.