Department of Computer Science & Engineering

University of Asia Pacific

Course Code: CSE 406

Course Title: Operating System Lab

Experiment No: 01

Submitted by

Name:Tamanna Akter Toma

Reg No: 21201074

Sec: B(B1)

Submitted to

Atia Rahman Orthi

Lecturer

Department of CSE,UAP

Submission Date: 13/02/25

Problem Statement:

Input:

Process	Arrival	Burst
P0	2	5
P1	0	3
P2	4	4

Output:

Process	Completion Time	Total Arrival Time	Waiting Time
P0	8	6	3
P1	5	5	0
P2	12	8	4

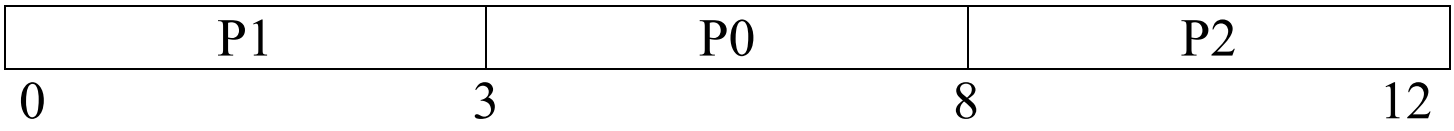
Steps of the (FCFS) algorithm:

1. All processes are placed in a queue in the order they arrive.
2. The CPU picks the first process in the queue and starts execution.
3. The process runs until it completes (non-preemptive scheduling).
4. Once the process finishes execution, it is removed from the queue.
5. The next process in the queue is selected and executed.
6. The steps repeat until all processes are executed.

Example:

Process	Arrival Time	Burst Time
P0	2	5
P1	0	3
P2	4	4

Gantt Chart Representation:



Turnaround Time (TAT) & Waiting Time (WT) Calculation:

- Turnaround Time (TAT) = Completion Time - Arrival Time
- Waiting Time (WT) = Turnaround Time - Burst Time

Process	**Completion Time**	**Total Arrival Time**	**Waiting Time**
P0	8	6	3
P1	5	5	0
P2	12	8	4

Source Code:

D: > Lab 01 > C fcfs.c > main()

```c
#include <stdio.h>

int main() {
    int arr1[] = {2, 0, 4}; // Arrival times
    int arr2[] = {5, 3, 4}; // Burst times
    int n = sizeof(arr1) / sizeof(arr1[0]);

    // Sort arrival times and adjust burst times accordingly
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr1[j] > arr1[j + 1]) {
                int temp = arr1[j];
                arr1[j] = arr1[j + 1];
                arr1[j + 1] = temp;

                temp = arr2[j];
                arr2[j] = arr2[j + 1];
                arr2[j + 1] = temp;
            }
        }
    }

    // Swap first two elements of burst time array
    int temp = arr2[0];
    arr2[0] = arr2[1];
    arr2[1] = temp;

    int completion[n], turnaround[n], waiting[n];

    // Calculate Completion Time (CT)
    completion[0] = arr1[0] + arr2[0];
    for (int i = 1; i < n; i++) {
        if (completion[i - 1] < arr1[i]) {
            completion[i] = arr1[i] + arr2[i];
        } else {
            completion[i] = completion[i - 1] + arr2[i];
        }
```

D: > Lab 01 > C fcfs.c > main()

```c
int main() {
    for (int i = 1; i < n; i++) {
        } else {
    }

    // Calculate Turnaround Time (TAT) and Waiting Time (WT)
    for (int i = 0; i < n; i++) {
        turnaround[i] = completion[i] - arr1[i];
        waiting[i] = turnaround[i] - arr2[i];
    }

    // Print results
    printf("Sorted Arrival Times: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr1[i]);
    }
    printf("\n");

    printf("Swapped Burst Times: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr2[i]);
    }
    printf("\n");

    printf("Completion Time: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", completion[i]);
    }
    printf("\n");

    printf("Turnaround Time: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", turnaround[i]);
    }
    printf("\n");

    printf("Waiting Time: ");
```

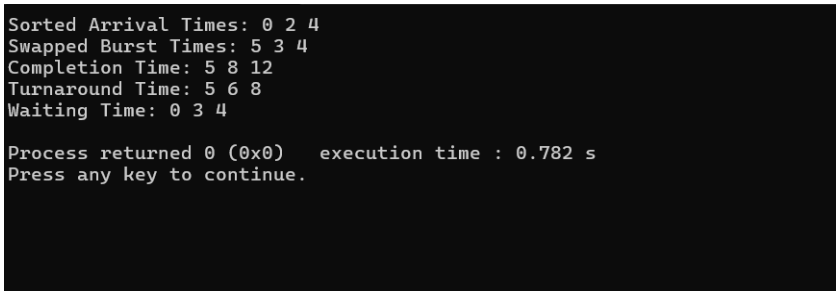
D: > Lab 01 > C fcfs.c > main()

```c
int main() {

    printf("Waiting Time: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", waiting[i]);
    }
    printf("\n");

    return 0;
}
```

Drive Link:

https://drive.google.com/file/d/1a3B9bv1UmhFvJJuLI32JHzw4BMv_SU3Q/view?usp=drive_link

Consoling Output:



Conclution:

1. **Sorting by Arrival Time** ensures proper scheduling.
2. **Swaps Burst Times** of the first two processes.
3. **Calculates Completion Time** considering CPU idle times.
4. **Derives Turnaround & Waiting Times** using completion and burst times.
5. **Displays Final Output** with sorted and modified values.
6. **Simulates Basic Scheduling** but doesn't follow a standard algorithm.