Operating Systems Lab
Fall 2024-25(L59+60)
Student Name:- Parth Suri
Registration No:- 22BDS0116
Class No. :- VL2024250102445
Faculty Name:- Rahul Srivastava

**Lab Manual: Implementing FCFS (First-Come, First-Served) Scheduling in Python**

**Objective**: To understand and implement the First-Come, First-Served (FCFS) scheduling algorithm using Python, taking into account process arrival times and burst times.

**Theory:** FCFS is one of the simplest scheduling algorithms. In FCFS, the process that arrives first is executed first. The processes are managed in a FIFO (First In, First Out) queue.

**Resources Required:**
• Python installed on your computer
• Text editor or Python IDE (e.g., VS Code, PyCharm, Jupyter Notebook)

**Experiment Steps:**

**1. Problem Definition:**
Define the problem by setting up a list of processes, each with specific arrival and burst times. The goal is to display the order of scheduling and calculate the completion time, turnaround time, and waiting time for each process using the FCFS scheduling algorithm.

**2. Initialize Data Structures:**
• Create a list to hold process IDs.
• Create a list to hold arrival times.
• Create a list to hold burst times.

**3. Input Data:**
Prompt the user to input the number of processes and their respective arrival and burst times.

**4. Sort Processes by Arrival Time:**
Sort the processes based on their arrival times to simulate FCFS scheduling.

**5. Calculate Completion Time:**
• Initialize a list to store completion times for each process.
• The completion time for each process is the sum of its burst time and the maximum of its arrival time and the completion time of the previous process.

**6. Calculate Turnaround Time:**
• Initialize a list to store turnaround times for each process.
• The turnaround time for each process is the difference between its completion time and its arrival time.

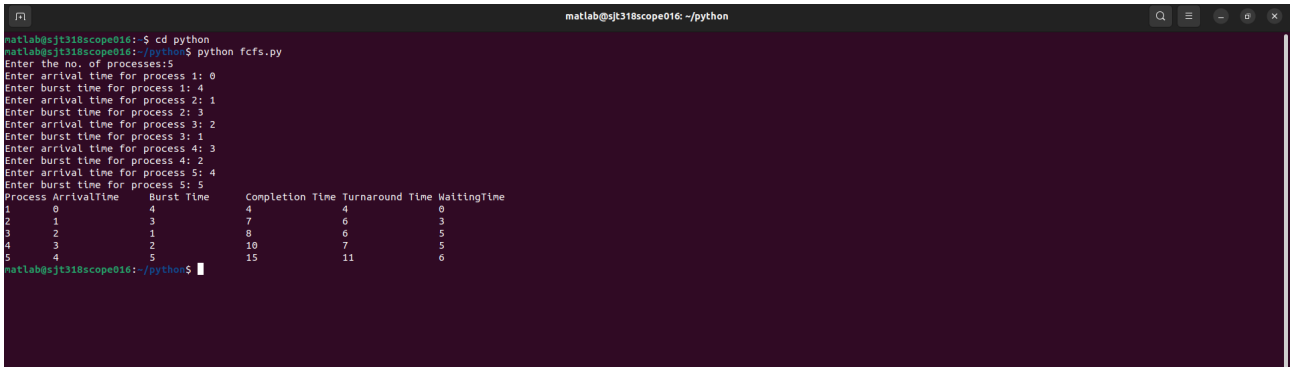
**7. Calculate Waiting Time:**
• Initialize a list to store waiting times for each process.
• The waiting time for each process is the difference between its turnaround time and its burst time.

**8. Display the Results:**
• Print the order in which processes are scheduled, their completion times, turnaround times, and waiting times.

**Code:**

```
#WAP for implementing FCFS(First Cum First Serve)
#code by PARTH SURI - 22BDS0116
no_of_processes = int(input("Enter the no. of processes:"))
process = []
arrival_time = [] #To store arrival time
burst_time = [] #To store burst time

for i in range(no_of_processes):
    process.append(i+1) #Process ID
    arrival_time.append(int(input(f"Enter arrival time for process {i+1}: ")))
    burst_time.append(int(input(f"Enter burst time for process {i+1}: ")))

process_info = list(zip(process,arrival_time,burst_time))
process_info.sort(key=lambda x: x[1]) #sorting by arrival time
process, arrival_time, burst_time = zip(*process_info)

completion_time = [0] * no_of_processes #Initialization of CT

for i in range(no_of_processes):
    if i == 0:
        completion_time[i] = arrival_time[i] + burst_time[i]
    else:
        completion_time[i] = max(completion_time[i-1] + burst_time[i], arrival_time[i] +
burst_time[i])

turnaround_time = [0] * no_of_processes # Initialization of TAT

for i in range(no_of_processes):
    turnaround_time[i] = completion_time[i] - arrival_time[i]

waiting_time = [0] * no_of_processes # Initialization of WT

for i in range(no_of_processes):
    waiting_time[i] = turnaround_time[i] - burst_time[i]

print("Process\tArrivalTime\tBurst Time\tCompletion Time\tTurnaround Time\tWaitingTime")
for i in range(no_of_processes):

    print(f"{process[i]}\t{arrival_time[i]}\t\t{burst_time[i]}\t\t{completion_time[i]}\t\
t{turnaround_time[i]}\t\t{waiting_time[i]}")
```

**Test Case 1:**




**Gantt Chart:**

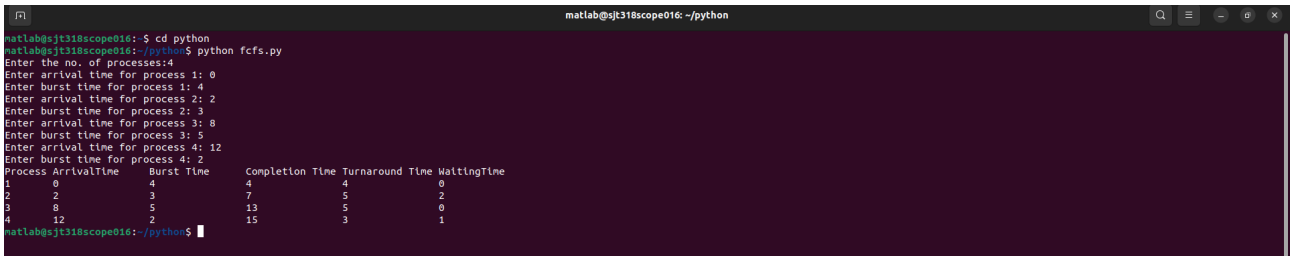
| P1 | P2 | P3 | P4 | P5 |

0 4 7 8 10 15

**Further Experimentation:**

• Demonstrate the Convoy Effect and highlight the disadvantage of the FCFS.
• The First-Come, First-Served (FCFS) scheduling algorithm schedules processes in the order they arrive. However, when processes arrive at different times, the CPU may be idle if no process is ready to execute. This variation of FCFS accounts for such idle times, ensuring correct scheduling and accurate calculation of completion, turnaround, and waiting times. Modify your code accommodate the above case and test your code over the following test case.

**Test case 2:**




**Gantt Chart:**

| P1 | P2 | Idle | P3 | P4 |

0 4 7 8 13 15

**Questions:**

1. What are the main disadvantages of the FCFS scheduling algorithm?

Main advantages are:-

1. **Long Average Waiting Time**: Since FCFS processes tasks in the order of their arrival, a single long process can cause subsequent processes to wait a long time.

2. **Convoy Effect:** Shorter processes may get stuck waiting behind longer ones, leading to inefficiencies and increased waiting times.

3. **Non-preemptive Nature:** Once a process starts, it cannot be interrupted until it finishes. This can be problematic for systems that need to respond quickly to important tasks.

4. **Poor Performance in Real-Time Systems:** FCFS is not suitable for real-time systems where timely processing is crucial.

2. How does the FCFS algorithm handle processes with the same arrival time?

In FCFS, if multiple processes have the same arrival time, they are scheduled in the order they are listed in the queue. There is no inherent mechanism to differentiate between them based on priority or other factors.

3. Can FCFS result in the starvation of processes? Why or why not?

No, FCFS does not lead to starvation. In FCFS, every process will eventually be executed since they are processed in the order they arrive. However, it can lead to long waiting times for shorter processes if a long process arrives before them.

4. How would you modify the algorithm to handle priority scheduling?

To modify FCFS for priority scheduling, you can introduce the following changes:

1. **Priority Queue:** Instead of a simple queue, use a priority queue where processes are enqueued based on their priority.

2. **Priority-Based Selection:** When selecting the next process to run, choose the one with the highest priority (or lowest priority number, depending on the convention).

3. **Preemptive Priority Scheduling:** To further refine the approach, make the scheduling preemptive. If a process with a higher priority arrives while a lower-priority process is running, the current process can be preempted and the higher-priority process can be executed.

This would make the scheduling more responsive to important tasks but could potentially reintroduce the risk of starvation for lower-priority processes, which might be addressed with techniques like aging (gradually increasing the priority of waiting processes).