Experiment No : 01

Date of Experiment : 22 / 02 / 2025

Experiment Name : First Come First Serve (FCFS) Scheduling

Theory: The full form of FCFS Scheduling is First Come First Serve Scheduling **.** FCFS Scheduling algorithm automatically executes the queued processes and requests in the order of their arrival . It is managed with a FIFO queue. The processes that request the CPU first get the allocation from the CPU first. So no project gets paused . Its Process Control Block (PCB) gets linked with the queue's tail. Even if the CPU starts working on a longer job, many shorter ones have to wait after it. The FCFS scheduling algorithm works in most of the batches of operating systems. FCFS is a very fair algorithm since no priority is involved . FCFS doesn't lead to any starvation.

Code:

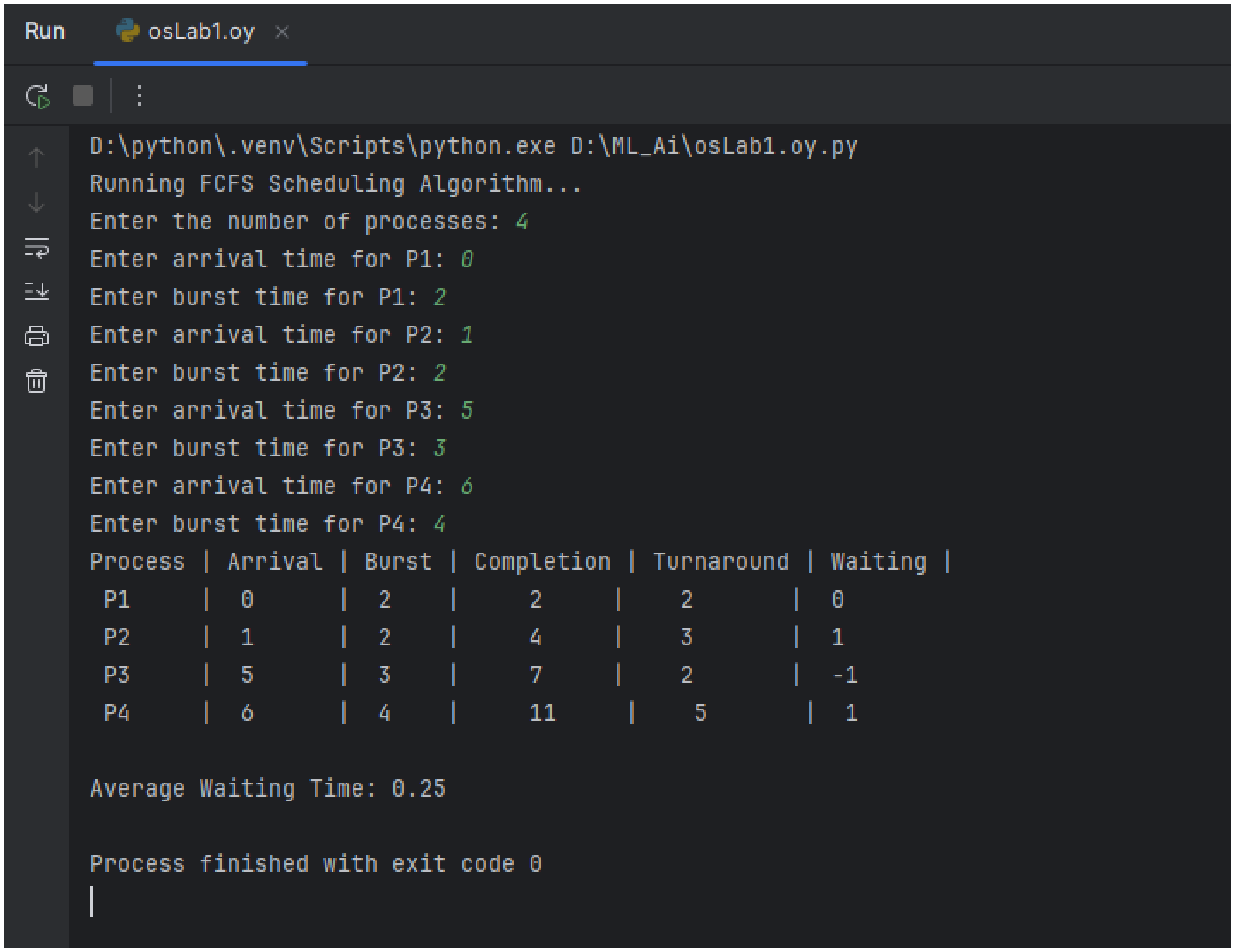
```
print("Running FCFS Scheduling Algorithm...")
n = int(input("Enter the number of processes: "))
d = {}
# Input arrival and burst times for each process
for i in range(n):
    key = "P" + str(i + 1)
    arrival = int(input(f"Enter arrival time for {key}: "))
    burst = int(input(f"Enter burst time for {key}: "))
    d[key] = [arrival, burst]
# Sort processes by arrival time (in case they are not already sorted)
d = sorted(d.items(), key=lambda item: item[1][0])

# Calculate Completion Time (CT) for each process
CT = []
for i in range(len(d)):
    CT.append(d[i][1][1] if i == 0 else CT[i - 1] + d[i][1][1])

# Calculate Turnaround Time (TAT) and Waiting Time (WT) for each process
TAT = [CT[i] - d[i][1][0] for i in range(len(d))]
WT = [TAT[i] - d[i][1][1] for i in range(len(d))]

# Calculate average waiting time
avg_WT = sum(WT) / n
# Print results in a table format
print("Process | Arrival | Burst | Completion | Turnaround | Waiting |")
for i in range(n):
    print(f" {d[i][0]}    | {d[i][1][0]}    | {d[i][1][1]}   |    {CT[i]}    |    {TAT[i]}     | {WT[i]}    ")
print(f"\nAverage Waiting Time: {avg_WT:.2f}")
```

Input and Output:



```
D:\python\.venv\Scripts\python.exe D:\ML_Ai\osLab1.oy.py
Running FCFS Scheduling Algorithm...
Enter the number of processes: 4
Enter arrival time for P1: 0
Enter burst time for P1: 2
Enter arrival time for P2: 1
Enter burst time for P2: 2
Enter arrival time for P3: 5
Enter burst time for P3: 3
Enter arrival time for P4: 6
Enter burst time for P4: 4
Process | Arrival | Burst | Completion | Turnaround | Waiting |
 P1     |  0      |  2    |     2      |    2       |  0
 P2     |  1      |  2    |     4      |    3       |  1
 P3     |  5      |  3    |     7      |    2       |  -1
 P4     |  6      |  4    |     11      |    5       |  1

Average Waiting Time: 0.25

Process finished with exit code 0
```

Explanation :

- **Input Handling:** The user enter the number of processes, arrival time, and burst time for each process.
- **Sorting**: Processes are sorted based on their arrival times . It used Python's sorted() function with a lambda function as the key.
- **Completion Time (CT):** Calculated iteratively by adding burst times, starting from the first process.
- **Turnaround Time (TAT):** Calculated as the difference between completion time and arrival time.
- **Waiting Time (WT):** Calculated as the difference between turnaround time and burst time.
- **Output:** Prints a table displaying process details along with completion time, turnaround time, and waiting time. It calculates and prints the average waiting time for all processes.

Discussion : The FCFS scheduling algorithm is non-preemptive and easy to implement . It has some Drawbacks of FCFS**:**

1. **Convoy Effect**: Longer processes can lead to a situation where shorter processes are delayed significantly (convoy effect), impacting overall system throughput.
2. **Non-optimal Average Waiting Time**: FCFS may result in higher average waiting times compared to other scheduling algorithms like Shortest Job Next (SJN) or Shortest Remaining Time (SRT), where shorter jobs are prioritized.
3. **No Priority Consideration**: FCFS treats all processes equally without considering their priority or importance, which may not be suitable for systems requiring different levels of service.
4. **Poor Utilization**: FCFS may not make optimal use of CPU resources, as it doesn't consider the burst time or remaining time of processes. This can lead to inefficient scheduling in terms of overall system performance.

Conclusion: FCFS scheduling is a fundamental CPU scheduling algorithm that is easy to understand and implement. However, it is not the most efficient in terms of minimizing waiting time and turnaround time. It works best in situations where process execution time is predictable and fairness is required. Future improvements in scheduling algorithms, such as priority-based or shortest-job-first scheduling, can help mitigate some of the inefficiencies seen in FCFS. Overall, FCFS remains a crucial concept in operating system design and process scheduling strategies.