Experiment No : 02
Date of Experiment : 20 / 04 / 2025
Experiment Name : Priority Scheduling

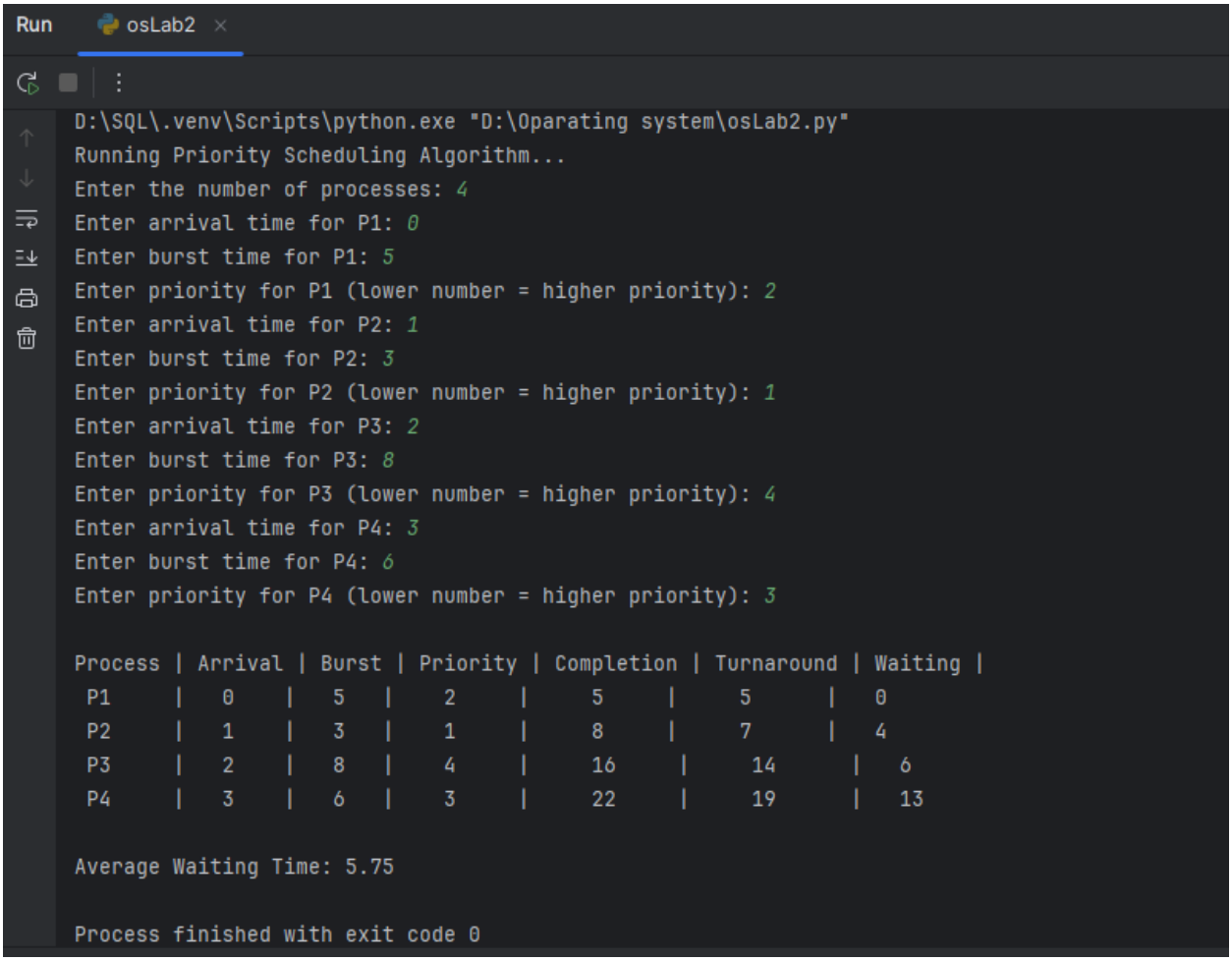
Theory:

Priority Scheduling is a CPU scheduling algorithm that assigns a priority to each process. The CPU is allocated to the process with the highest priority (usually the lowest numerical value represents the highest priority). If two processes have the same priority, they are scheduled according to their arrival order. This algorithm can be either **preemptive** or **non-preemptive**, depending on whether a process can be interrupted by a newly arrived process with higher priority.

Code:

```
print("Running Priority Scheduling Algorithm...")
n = int(input("Enter the number of processes: "))
d = {}
# Input arrival time, burst time, and priority 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}: "))
    priority = int(input(f"Enter priority for {key} (lower number = higher priority): "))
    d[key] = [arrival, burst, priority]
# Sort based on arrival time and then priority
d = sorted(d.items(), key=lambda item: (item[1][0], item[1][2]))
# Calculate Completion Time (CT)
CT = []
current_time = 0
for i in range(len(d)):
    arrival, burst, _ = d[i][1]
    if current_time < arrival:
        current_time = arrival
    current_time += burst
    CT.append(current_time)
# Calculate Turnaround Time (TAT) and Waiting Time (WT)
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))]
# Average Waiting Time
avg_WT = sum(WT) / n
print("\nProcess | Arrival | Burst | Priority | Completion | Turnaround | Waiting |")
for i in range(n):
    print(f" {d[i][0]}    |   {d[i][1][0]}   |  {d[i][1][1]}  |   {d[i][1][2]}    |    {CT[i]}    |    {TAT[i]}     |   {WT[i]}")
print(f"\nAverage Waiting Time: {avg_WT:.2f}")
```

Input and Output:



Explanation:

- **Input Handling:** The user inputs the number of processes, along with each process's arrival time, burst time, and priority.
- **Sorting:** Processes are sorted first by arrival time and then by priority. Lower numerical priority values mean higher importance.
- **Completion Time (CT):** Computed by simulating process execution from earliest arrival, taking CPU idle time into account if needed.
- **Turnaround Time (TAT):** The time from process arrival to its completion (CT - Arrival).
- **Waiting Time (WT):** The time a process spends waiting in the queue (TAT - Burst).
- **Output:** A formatted table showing all process metrics and average waiting time.

Discussion:

Priority Scheduling is useful in situations where some processes are more important than others. However, it has its own drawbacks:

1. **Starvation:** A continuous inflow of high-priority processes can indefinitely delay lower-priority ones.
2. **Complexity:** More complex than FCFS due to sorting based on priority and arrival.
3. **Fairness:** Unlike FCFS, fairness is not guaranteed since low-priority processes can suffer long delays.
4. **Scheduling Overhead:** Frequent comparison of process priorities adds overhead, especially in dynamic systems.

Conclusion:

Priority Scheduling enhances process management by allowing important tasks to execute first. However, it introduces complexity and fairness issues like starvation. Aging (gradually increasing the priority of waiting processes) is a common technique to address starvation. This algorithm is more efficient than FCFS in many real-time and interactive systems, especially when tasks have different levels of importance.