Department of Computer Science & Engineering

Course Title: Operating System Lab

Course Code: CSE 406

Lab Report No: 05

Lab Report: Priority Scheduling

Submission Date: 13-03-2025

Submitted To:

Atia Rahman Orthi

Lecturer,

Department of CSE, UAP

Submitted By:

Name: Amirul Islam Papon

Reg No: 21201076

Sec: B

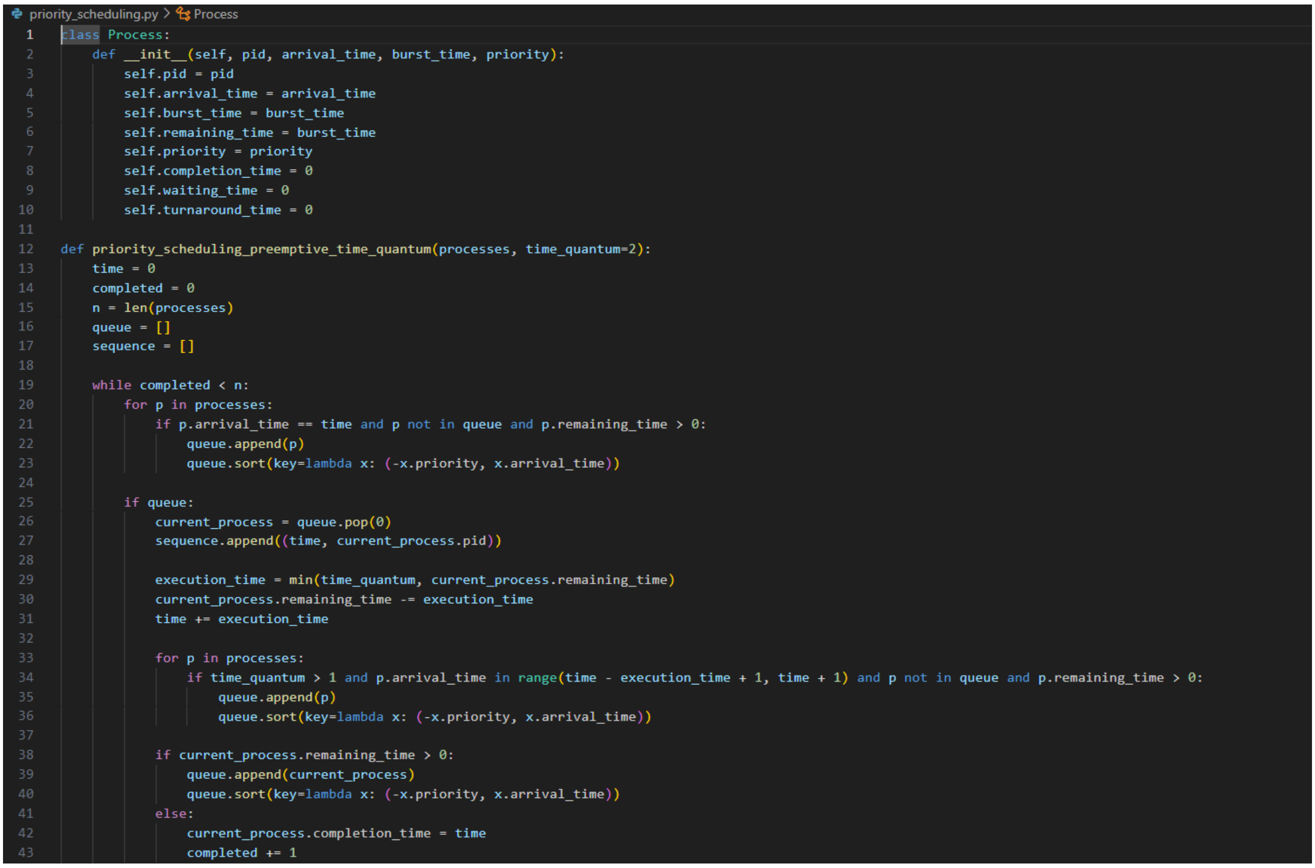
Priority Scheduling

Problem Statement: Priority Scheduling is a CPU scheduling algorithm where each process is assigned a priority, and the process with the highest priority is executed first. In the case of preemptive priority scheduling, a higher-priority process can preempt the currently running process. This report discusses the implementation of preemptive priority scheduling with a time quantum, ensuring fair execution among processes while maintaining priority constraints.

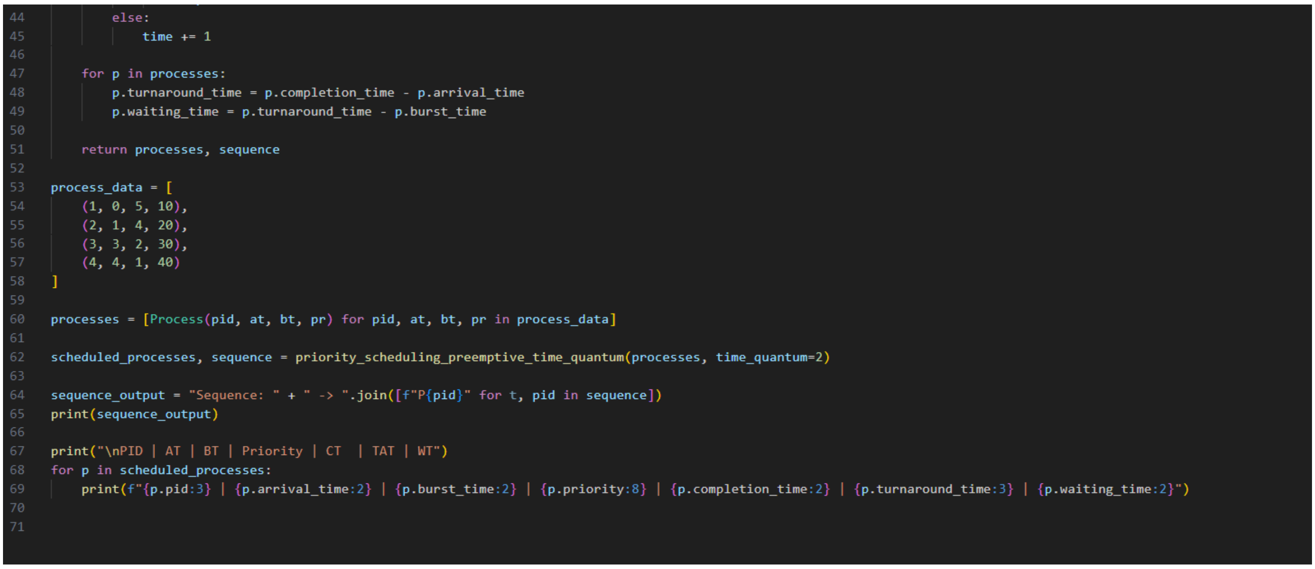
Steps:

1. **Initialize Process Data:**
   - Define each process with its Process ID (PID), Arrival Time (AT), Burst Time (BT), and Priority.
2. **Sort and Manage Execution:**
   - At each time step, select the highest priority process that has arrived.
   - Execute the selected process for a predefined time quantum.
   - If the process completes within the quantum, mark it as completed.
   - If not, reinsert it into the queue for future execution.
3. **Preempt When Necessary:**
   - If a new process with a higher priority arrives, preempt the currently running process.
   - Add the preempted process back to the queue with its remaining burst time.
4. **Continue Until All Processes Complete:**
   - Keep scheduling processes until all are executed completely.
   - Maintain a Gantt chart to track execution order.
5. **Calculate Performance Metrics:**
   - Compute Completion Time (CT), Turnaround Time (TAT = CT - AT), and Waiting Time (WT = TAT - BT).

Source Code:

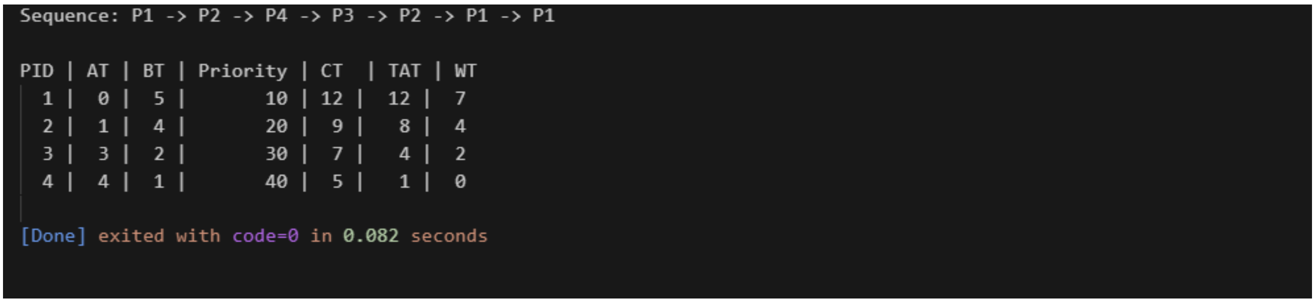


```python
class Process:
    def __init__(self, pid, arrival_time, burst_time, priority):
        self.pid = pid
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.remaining_time = burst_time
        self.priority = priority
        self.completion_time = 0
        self.waiting_time = 0
        self.turnaround_time = 0

def priority_scheduling_preemptive_time_quantum(processes, time_quantum=2):
    time = 0
    completed = 0
    n = len(processes)
    queue = []
    sequence = []

    while completed < n:
        for p in processes:
            if p.arrival_time == time and p not in queue and p.remaining_time > 0:
                queue.append(p)
                queue.sort(key=lambda x: (-x.priority, x.arrival_time))

        if queue:
            current_process = queue.pop(0)
            sequence.append((time, current_process.pid))

            execution_time = min(time_quantum, current_process.remaining_time)
            current_process.remaining_time -= execution_time
            time += execution_time

            for p in processes:
                if time_quantum > 1 and p.arrival_time in range(time - execution_time + 1, time + 1) and p not in queue and p.remaining_time > 0:
                    queue.append(p)
                    queue.sort(key=lambda x: (-x.priority, x.arrival_time))

            if current_process.remaining_time > 0:
                queue.append(current_process)
                queue.sort(key=lambda x: (-x.priority, x.arrival_time))
            else:
                current_process.completion_time = time
                completed += 1
```



```python
        else:
            time += 1

    for p in processes:
        p.turnaround_time = p.completion_time - p.arrival_time
        p.waiting_time = p.turnaround_time - p.burst_time

    return processes, sequence

process_data = [
    (1, 0, 5, 10),
    (2, 1, 4, 20),
    (3, 3, 2, 30),
    (4, 4, 1, 40)
]

processes = [Process(pid, at, bt, pr) for pid, at, bt, pr in process_data]

scheduled_processes, sequence = priority_scheduling_preemptive_time_quantum(processes, time_quantum=2)

sequence_output = "Sequence: " + " -> ".join([f"P{pid}" for t, pid in sequence])
print(sequence_output)

print("\nPID | AT | BT | Priority | CT  | TAT | WT")
for p in scheduled_processes:
    print(f"{p.pid:3} | {p.arrival_time:2} | {p.burst_time:2} | {p.priority:8} | {p.completion_time:2} | {p.turnaround_time:3} | {p.waiting_time:2}")
```

Output:



```
Sequence: P1 -> P2 -> P4 -> P3 -> P2 -> P1 -> P1

PID | AT | BT | Priority | CT  | TAT | WT
  1 |  0 |  5 |       10 | 12 |  12 |  7
  2 |  1 |  4 |       20 |  9 |   8 |  4
  3 |  3 |  2 |       30 |  7 |   4 |  2
  4 |  4 |  1 |       40 |  5 |   1 |  0

[Done] exited with code=0 in 0.082 seconds
```

Discussion & Conclusion: Preemptive Priority Scheduling ensures that high-priority processes are executed first, reducing response time for critical tasks. However, lower-priority processes may suffer from starvation if higher-priority processes keep arriving.

Source Code: https://github.com/Amirul-Islam-Papon/Operating-System/blob/main/priority_scheduling.py