Sauhard Shakya

005031397

 Advanced Operating Systems (MSCS-630-A01)

Project Deliverable 2: Process Scheduling

**Abstract**

This deliverable extends the custom Java shell developed in Phase 1 by adding a simulation of process scheduling algorithms. It demonstrates how an operating system allocates CPU time to processes using Round-Robin scheduling and Priority-based scheduling. Each simulated process has attributes such as burst time, priority, and arrival time, and the scheduling system calculates important performance metrics such as waiting time, turnaround time, and response time. By simulating these algorithms inside the shell, the project models real-world operating system behavior in a simplified and controlled way.

**1. Overview**

The purpose of Deliverable 2 is to integrate CPU scheduling into the custom shell so that it does more than simply execute commands. In this phase, the shell uses simulated processes to represent tasks competing for CPU time. Each process is stored as a data structure with fields such as process ID, burst time, remaining time, priority, and arrival time. By running these simulated processes under different scheduling algorithms, the shell can demonstrate how an operating system decides which process should be executed at a given moment. This phase moves the focus from process management at the shell level to process scheduling at the CPU level.

**2. Scheduling Algorithms**

Two scheduling algorithms are implemented. The first is Round-Robin scheduling, which is designed to ensure fairness. Each process is given a fixed time slice, also known as a quantum. If the process does not finish within that time, it is placed at the back of the queue and waits for another turn. This ensures that no single process can monopolize the CPU and that every process continues to make progress. For example, with a quantum of 2, each process runs for 2 ticks before being swapped out if it still needs more time.

The second algorithm is Priority-based scheduling. In this method, each process is assigned a priority number, and the scheduler always selects the highest-priority process to run first. If two processes have the same priority, the scheduler uses a First-Come, First-Served policy to decide which one goes first. Priority scheduling also includes preemption: if a new process with higher priority arrives while a lower-priority process is running, the scheduler interrupts the current process and immediately gives control to the higher-priority one. This allows critical tasks to be handled as soon as they arrive but may cause starvation for low-priority tasks.

**3. Implementation Details**

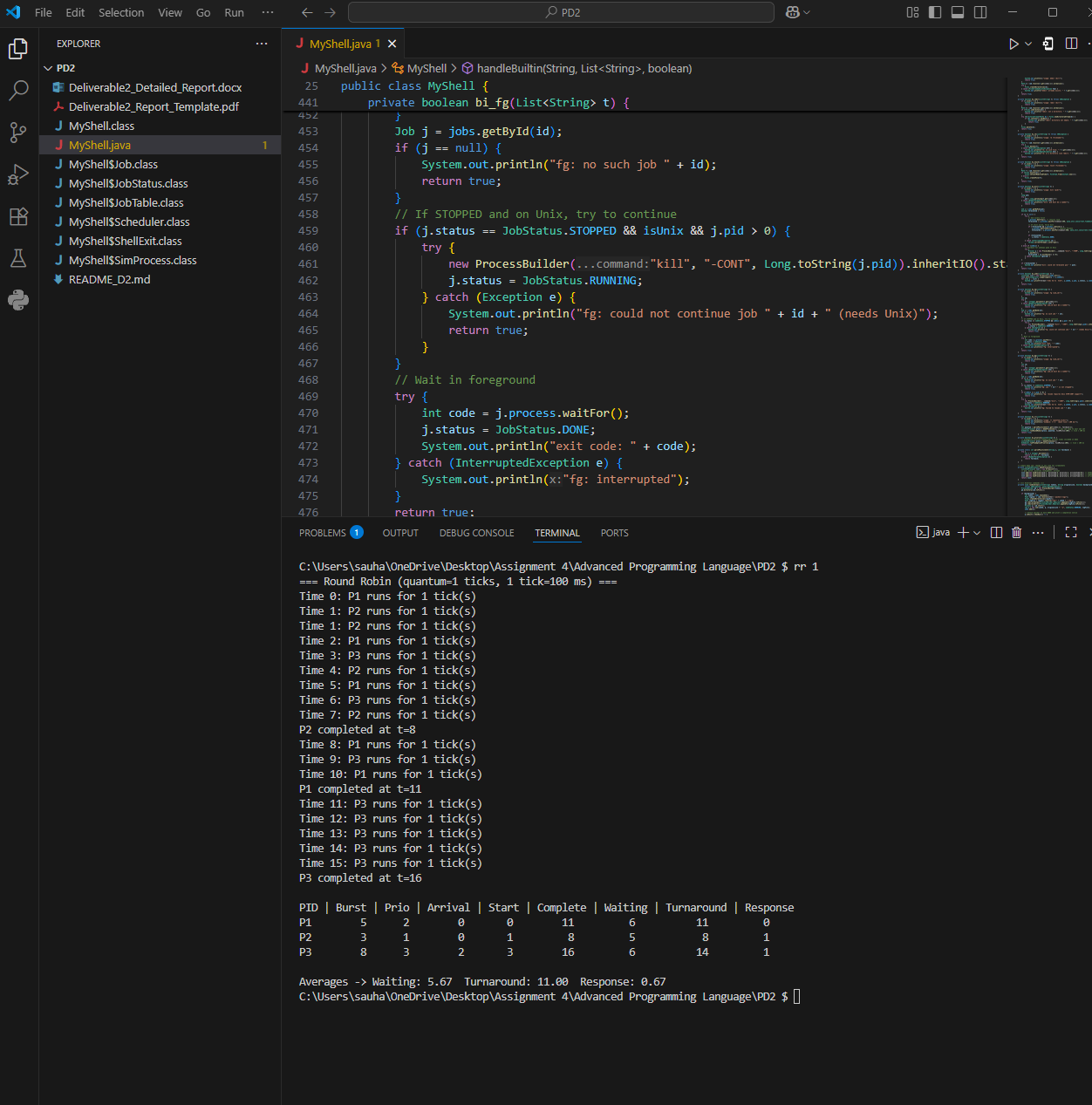
The scheduling simulation is implemented using two classes: SimProcess and Scheduler. SimProcess represents a process control block and includes attributes such as process ID, burst time, priority, arrival time, start time, and completion time. The Scheduler class implements the logic for both Round-Robin and Priority scheduling. Round-Robin uses a queue to rotate processes based on the time quantum, while Priority scheduling uses a priority queue to always select the process with the highest priority. The simulation uses short delays to represent time passing so that the scheduling decisions can be observed directly in the console output.

**4. Performance Metrics**

The performance of the scheduling algorithms is evaluated using three common metrics: waiting time, turnaround time, and response time. Waiting time is the amount of time a process spends in the ready queue before it gets CPU time. It is calculated as turnaround time minus burst time. Turnaround time is the total time taken for a process to complete, measured from its arrival until its completion. Response time is the time between a process’s arrival and the moment it first gets CPU time. These metrics help measure the efficiency of the scheduling algorithm. For example, Round-Robin tends to improve response time because processes are given CPU time quickly, while Priority scheduling improves waiting and turnaround times for high-priority tasks but may increase them for lower-priority ones.

**5. Screenshots**

Screenshots demonstrate the execution of Round-Robin scheduling with different time slices to show how processes are rotated. Screenshots also show Priority scheduling in action, including cases where a high-priority process preempts a lower-priority process. Finally, screenshots include the metrics tables printed by the shell after each simulation, showing the calculated waiting, turnaround, and response times for all processes and their averages.



A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**6. Challenges and Improvements**

One of the main challenges in this deliverable was simulating process scheduling inside Java. Unlike a real operating system, Java does not allow direct control over hardware scheduling or system-level interrupts. To address this, the simulation was implemented using small time ticks and queues to mimic how scheduling works. Another challenge was ensuring that preemption worked correctly in Priority scheduling, so that a higher-priority process arriving mid-execution would take over immediately.

Future improvements could include implementing additional algorithms such as Shortest Job First (SJF), Multi-Level Feedback Queues, or adding an aging mechanism to prevent starvation in Priority scheduling. A graphical Gantt chart representation could also be added to make the scheduling order and process switching easier to visualize.