Java Multithreading CPU Scheduler Report

CSC227: OPERATING SYSTEMS

Computer Science Department — Spring Semester 2025

King Saud University — Project Report

Team Members

Mohannad alshahrani	442100744
Saud Hamad Alrayes	443170180
Yasser Kalled Al-Qasem	444103067

Objective

To implement a multithreaded Java program that simulates a CPU scheduler supporting three scheduling algorithms:

- First-Come-First-Serve (FCFS)
- Round Robin (RR, Quantum = 7ms)
- Priority Scheduling

Assumption:

1. In our simulation, a job is considered to have suffered from **starvation** if its **waiting time exceeds the average waiting time** of all executed jobs.

2. We begin counting starvation time from the moment a job is **added to the ready queue**, not when it's initially loaded from the file.

Threads

- **JobLoader Thread:** Reads job.txt, constructs PCBs, and loads them into a job queue.
- **Scheduler Thread:** Fills ready queue then Executes the selected scheduling algorithm, manages memory, and logs execution.

multithreading improved responsiveness: job loading doesn't block execution, and memory allocation decisions happen concurrently. This simulates how real-world OS components run independently.

Classes Overview

- Job: Represents a process with fields like burst time, memory, priority, etc.
- MemoryManager: Manages allocation and deallocation of memory.
- Scheduler (abstract): Base class for all scheduling strategies.
  - **FCFSScheduler:**
    - Jobs are executed in the order they arrive.
  - **RoundRobin:**
    - Quantum: 7ms
    - Preemptive algorithm that cycles through jobs.
  - **Priority:**
    - Jobs scheduled by priority (1 = lowest, 8 = highest)
    - Custom priority queue implementation

System Calls Simulated

Simulated Call	**Arguments**	**Output/Effect**
allocateMemory(size)	int size	Returns true if memory available; false otherwise
deallocateMemory(size)	int size	Frees memory used by a job

Output Representation Preference

We preferred Gantt chart format for visual clarity. It makes execution order and timing **in console**

Sample Output:

```
Gantt Chart:

|0 P3 |22 P6 |34 P1 |52 P4 |67 P7 |92 P5 |122 P9 |136 P10 |156 P2 |166 P8 |174


Results:

Average Waiting Time: 37.80 ms

Average Turnaround Time: 55.20 ms
```

(Starved Jobs appears only in priority)

```
Starved Jobs Detected:

P4 | Waited: 52ms

P2 | Waited: 156ms

P8 | Waited: 74ms
```

Software and Hardware Tools Used

- **Language:** Java 17+
- **IDE:** Visual Studio Code
- **Operating System:** Windows 11
- **GitHub:** Version control and collaboration

Program Strengths

- extensible class design
- Clear Gantt chart output and average metrics
- Memory handling works independently via threads
- Starvation detection in Priority Scheduling implemented dynamically, adaptive starvation handling

Program Weaknesses

- No GUI (console-only)

How to Extend to Full OS Simulation

To simulate a full OS, the program could be extended by:

- Adding **I/O management**
- Introducing **context switch delay simulation**
- Supporting **process creation/termination** during execution
- Including **paging** or **virtual memory simulation**
- Simulating **multi-core CPUs** with more scheduler threads

Attachments

- Source code (GitHub): https://github.com/mths0/Java-Multithreading-Project
- Sample job.txt with 30 jobs
