Project Plan: Assignment 3 - CPU Scheduler Simulator

1.0 Project Overview: A Non-Technical Explanation

This document outlines the project plan for the development of a CPU Scheduler Simulator. Understanding CPU scheduling is a fundamental concept in computer science, essential for building efficient, responsive, and reliable operating systems. This project serves as a practical application of these core principles, moving from theoretical knowledge to a tangible software implementation that models how a central processing unit (CPU) allocates its resources among competing tasks.

1.1 The Core Challenge: Managing CPU Resources

At any given moment, a modern computer has numerous processes—from the operating system itself to user applications—all competing for the CPU's attention. CPU scheduling is the mechanism that decides which of these ready-to-run processes gets to use the CPU next and for how long. This is analogous to a manager assigning tasks to a single, highly efficient employee; the manager's strategy determines which tasks get done, in what order, and how quickly the overall workload is completed. The central problem scheduling solves is the fair and efficient allocation of the CPU's finite time to ensure the system remains responsive and productive.

1.2 Assignment Goal: Simulating and Comparing Schedulers

The primary objective of this project is to build a Java program that simulates four distinct methods, or schedulers, for managing CPU tasks. The goal is not just to implement these algorithms but to analyze and compare their performance. By running the same set of processes through each scheduler, we will measure their effectiveness using standard industry metrics, such as how long processes wait and how long they take to complete from start to finish.

1.3 The Four Scheduling Strategies

The simulation will implement the following four scheduling algorithms, each with a unique approach to managing processes:

- **Preemptive Shortest-Job First (SJF):** Focuses on completing the quickest tasks first to minimize overall average wait times.
- **Round Robin (RR):** Gives each task a small, equal slice of CPU time to ensure fairness and prevent any single process from monopolizing the system.
- **Preemptive Priority:** Gives preferential treatment to more important tasks, allowing them to interrupt less critical ones.

- **AG Scheduling:** A hybrid model that dynamically adjusts its strategy and a process's allocated time slice based on its runtime behavior.

The following sections will break down the precise rules and logic required to implement these schedulers and deliver a successful project.

2.0 Technical Specifications and Implementation Requirements

This section serves as the technical blueprint for the project. It details the precise algorithms, input/output formats, and performance metrics required for a successful implementation, as specified in the assignment document. Adherence to these specifications is critical for ensuring the simulator is accurate, testable, and meets all grading criteria.

2.1 Program Inputs and Core Parameters

The simulator must be configured at runtime with user-provided inputs. These parameters define the workload and the environment for the simulation.

- **Global Parameters:**
  - Number of processes
  - Round Robin Time Quantum
  - Context switching time
- **Per-Process Parameters:**
  - Process Name
  - Process Arrival Time
  - Process Burst Time
  - Process Priority
  - Process initial Quantum Time (for AG Scheduler)

2.2 Detailed Scheduler Logic

The core of the project is the implementation of the four distinct scheduling algorithms. Each must follow the specific logic outlined below.

2.2.1 Preemptive Shortest-Job First (SJF)

This scheduler prioritizes the process with the smallest remaining burst time. The implementation must be preemptive, meaning if a new process arrives in the ready queue with a shorter burst time than the currently running process, the running process must be interrupted, and the new, shorter process must be executed. All context switching overhead must be accounted for during these preemptions.

2.2.2 Round Robin (RR)

This scheduler implements a standard Round Robin algorithm. Each process in the ready queue is allowed to run for a time slice equal to the user-defined time quantum. If the process is not finished by the end of its time quantum, it is preempted and moved to the back of the ready queue. The next process at the front of the queue is then dispatched. The context switching time must be factored in after each time quantum expires.

2.2.3 Preemptive Priority Scheduling

In this model, processes are executed based on their priority level. A process with a higher priority will preempt a currently running process with a lower priority. A critical requirement for this implementation is to include a mechanism that solves the starvation problem, ensuring that low-priority processes are not indefinitely postponed and eventually get a chance to run.

Implementation Note: A common strategy to prevent starvation is 'aging,' where a process's priority is gradually increased the longer it remains in the ready queue. The team should implement this or a similar mechanism.

2.2.4 AG Scheduling Algorithm

This is the most complex algorithm, combining elements of four different scheduling disciplines (FCFS, Priority, SJF, and RR) with a dynamic quantum. Meticulous attention to the state transitions and quantum updates is essential for a correct implementation.

- **Multi-Stage Execution:** Within a single allocated time slice (its quantum), a process executes in up to three stages:
    - **0% to ceil(25%) of Quantum:** The process runs using First-Come, First-Served (FCFS).
    - **ceil(25%) to ceil(50%) of Quantum:** The process runs using non-preemptive Priority scheduling.
    - **ceil(50%) to 100% of Quantum:** The process runs using preemptive Shortest-Job First (SJF).
- **Dynamic Quantum Update Rules:** After a process leaves the running state (either by completing, being preempted, or finishing its quantum), its quantum time for the next turn is updated according to one of the following four scenarios:
    - **Scenario 1:** If the process utilized its entire quantum time and still has work to do, it is moved to the end of the ready queue, and its new quantum is increased by 2.
    - **Scenario 2:** If the process was preempted during its non-preemptive Priority stage (the second stage), it is moved to the end of the ready queue, and its new quantum is increased by ceil(remaining Quantum time / 2).
    - **Scenario 3:** If the process was preempted during its preemptive SJF stage (the final stage), it is moved to the end of the ready queue, and its new quantum is increased by the full remaining quantum time.

- **Scenario 4:** If the process finished its job before its quantum expired, its quantum value is set to 0, as it no longer requires CPU time.

2.3 Required Outputs and Performance Metrics

For each implemented scheduler, the program must calculate and display a standard set of performance metrics. The output must be clearly formatted and present the following information:

Metric	Requirement
Execution Order	Display the sequence of processes as they are executed by the CPU.
Waiting Time	Calculate and display the waiting time for each individual process.
Turnaround Time	Calculate and display the turnaround time for each individual process.
Average Waiting Time	Calculate and display the average waiting time across all processes.
Average Turnaround Time	Calculate and display the average turnaround time across all processes.
AG Quantum History	For the AG Scheduler ONLY, print all historical updates of the quantum time for each process.

2.4 Testing and Validation

A significant portion of the project grade depends on the implementation of robust unit tests. These tests must utilize the test cases provided with the assignment to rigorously validate the correctness of each scheduling algorithm's logic and its calculated outputs.

This comprehensive technical specification provides the foundation for development, which will be organized according to the project execution plan in the next section.

3.0 Task Distribution and Project Execution Plan

A clear division of labor is essential for the timely and successful completion of this project. This section assigns specific responsibilities to each team member—Hazem, Mohamed, and Omar—to ensure a balanced workload, establish clear ownership of components, and streamline the development and integration process.

3.1 Team Member Roles

Primary roles are assigned based on a logical division of the project's complexity and components:

- **Hazem:** Designated as the lead for the core architecture and the implementation of the most complex algorithm, the **AG Scheduler**. This role focuses on establishing a robust framework that all other components will build upon.
- **Mohamed:** Responsible for implementing the two standard scheduling algorithms, **Preemptive SJF** and **Round Robin (RR)**. Mohamed will also develop the final reporting module responsible for calculating and displaying all required output metrics.
- **Omar:** Tasked with implementing the **Preemptive Priority scheduler**, including its critical anti-starvation logic. Omar will also lead the crucial **integration and unit testing** efforts, ensuring all components work together correctly and meet the assignment's validation requirements.

3.2 Detailed Task Breakdown

The project is broken down into the following discrete tasks, with clear assignments, deliverables, and dependencies.

Task	Assigned To	Key Deliverables	Dependencies
1. Simulator Framework	Hazem	Core Java classes for Process, Scheduler interface, and the main simulation loop.	None
2. SJF & RR Schedulers	Mohamed	Two separate Java classes implementing the Scheduler interface for Preemptive SJF and Round Robin.	Task 1: Simulator Framework
3. Priority Scheduler	Omar	A Java class implementing the Scheduler interface for Preemptive Priority, including anti-starvation logic.	Task 1: Simulator Framework

4. AG Scheduler	Hazem	A Java class implementing the Scheduler interface for the complex AG Scheduling logic.	Task 1: Simulator Framework
5. Input/Output & Reporting	Mohamed	Code to parse all user inputs and format all required performance metrics for console output.	Tasks 2, 3, 4: Completed Schedulers
6. Unit Testing & Integration	Omar	A comprehensive suite of unit tests for all schedulers using provided test cases. The final, integrated, and runnable program.	Tasks 2, 3, 4, 5: All modules complete

The team is committed to following this plan to successfully deliver a robust, well-documented, and thoroughly tested CPU scheduler simulator.