WOLAITA SODO UNIVERSITY

SCHOOL OF INFORMATICS

DEPARTMENT OF : COMPUTER SCIENCE

COURSE TITLE : DATA STURCTURE AND ALGORITHIM

GROUP ASSIGNMENT

GROUP MEMBER AND ID NUMBER

NO	GROUP MEMBER	ID NO
1	KASU JIMA	Ugr/91890/16
2	Maidote birhan	Ugr/91947/16
3	Bekalu dems	Ugr/92553/16
4	Tsion zena	Ugr/92386/16
5	Bereket tilahun	Ugr/93801/16
6	Metages oroba	Ugr/92705/16

1. Introduction

This implements a job scheduling system that simulates three CPU scheduling algorithms: First-Come, First-Served (FCFS), Shortest Job First (SJF), and Round Robin (RR). It is designed to demonstrate how priority queues and other data structures can efficiently manage job execution while providing metrics like waiting time and turnaround time.

Key Features

Supports FCFS, SJF (non-preemptive), and RR scheduling.
Tracks job start/finish times and computes performance metrics.
Extensible design for adding new scheduling algorithms.
Clear console output for analyzing results.
Purpose: Represents a job with metadata for scheduling.

Fields

`id`: Unique job identifier.
`arrivalTime`: Time when the job arrives in the system.
`burstTime`: Total CPU time required.
`remainingTime`: Remaining time for RR scheduling.
`startTime/finishTime`: Track when the job starts and completes.

Methods

`addJob()`: Adds a job to the scheduler.

`schedule()`: Executes jobs according to the algorithm.
`printResults()`: Displays scheduling metrics.

Derived Scheduler Classes

`FCFSScheduler`

Uses a sorted vector to process jobs by arrival time.
`SJFScheduler`
Uses a priority queue (min-heap) to prioritize shorter jobs.
`RoundRobinScheduler`
Uses a FIFO queue with time slicing (quantum).

Approach

Job Structure: Define a `Job` structure to encapsulate job attributes such as ID, arrival time, burst time, remaining time, next available time (for RR), start time, and finish time.
Scheduler Classes: Create a base `Scheduler` class and derive three classes (`FCFSScheduler`, `SJFScheduler`, and `RoundRobinScheduler`) to implement each scheduling algorithm.
Priority Queues: Use C++ `priority_queue` with custom comparators to handle the ordering of jobs based on different criteria for each algorithm:
FCFS: Order jobs by their arrival time.
SJF: Order jobs by their burst time once they have arrived.
Round Robin: Order jobs by their next available time to simulate cyclic execution.
Scheduling Logic: Each derived class implements the scheduling logic specific to its algorithm, processing jobs and updating their states as they execute.

Algorithms

First-Come, First-Served (FCFS)

Logic: Jobs are executed in the order of their arrival.
Data Structure: Sorted vector (by arrival time).

Shortest Job First (SJF)

Logic: The job with the shortest burst time runs next.
Data Structure: Priority queue (min-heap ordered by burst time).

Round Robin (RR)

Logic: Jobs run in cycles with a fixed time quantum.

Data Structure: FIFO queue.

Performance Analysis and Improvements

1. First-Come, First-Served (FCFS)

Time Complexity:
Current Implementation:
Adding `n` jobs to a priority queue: `O(n log n)` (each insertion is `O(log n)`).
Extracting all jobs: `O(n log n)` (each extraction is `O(log n)`).
Total: `O(n log n)`.

Improved Approach:
Collect jobs in a vector, sort by arrival time: `O(n log n)`.
Process jobs in sorted order: `O(n)`.
Total: `O(n log n)` (same asymptotic complexity but fewer operations).
Space Complexity: `O(n)` (storing jobs).
Improvement: Replace the priority queue with a sorted vector to reduce constant factors.

2. Shortest Job First (SJF):

Time Complexity:
Sorting jobs by arrival time: `O(n log n)`.
Processing jobs with a priority queue (min-heap): Each job is inserted and extracted once, leading to `O(n log n)`.

Total: `O(n log n)`.

Space Complexity: `O(n)`.

Current Implementation: Optimal for non-preemptive SJF. No changes needed.

3. Round Robin (RR)

Current Implementation:
Uses a priority queue ordered by `nextAvailableTime`.
Each insertion and extraction: `O(log n)`.
For a job with burst time `B`, it is processed `O(B/quantum)` times.
Total Time: `O((ΣB_i/quantum) * log n)`, where `ΣB_i` is the sum of all burst times.
Space Complexity: `O(n)`.
Problem: Priority queue overcomplicates RR and increases time complexity.

Improved Approach

Use a standard FIFO queue and track job arrivals:
Sort jobs by arrival time: `O(n log n)`.
Process jobs in cycles: Each enqueue/dequeue is `O(1)`.
Total Time: `O(n log n + ΣB_i/quantum)` (asymptotically better)

Strengths of the Project

Efficient Data Structures
Priority Queues (for SJF/FCFS) and FIFO Queues (for RR) ensure optimal job ordering with minimal overhead.
Example: SJF's min-heap (`priority_queue`) efficiently retrieves the shortest job in `O(1)` time.

Modular and Extensible Design

The `Scheduler` base class allows easy addition of new algorithms (e.g., Priority Scheduling) without disrupting existing code.
Derived classes encapsulate algorithm-specific logic cleanly.
Correctness and Clarity
Implements scheduling algorithms faithfully (e.g., RR strictly enforces quantum time slicing).
Metrics like waiting time and turnaround time are tracked accurately.
Performance Optimizations
FCFS uses a sorted vector instead of a priority queue, reducing constant factors.
RR's FIFO queue ensures `O(1)` enqueue/dequeue operations during time slicing.
Readable Output
Results are formatted clearly for debugging and analysis (e.g., tables with job timelines).

Limitations and Areas for Improvement

Algorithm Restrictions
SJF is non-preemptive: Jobs cannot be interrupted if a shorter job arrives later.
No Priority Scheduling: The system does not support jobs with explicit priorities.
Static Job Input
Jobs are hardcoded in `main()`, limiting real-world usability.
Fix: Add a CLI or file input for dynamic job definitions.
Lack of Preemption
The SJF and FCFS implementations assume all jobs are available at arrival time.

Improvement: Add preemption for SJF (SRTF) or Priority Scheduling.

Scalability Concerns

Sorting jobs (`O(n log n)`) may become a bottleneck for extremely large datasets (e.g., 1M+ jobs).

Mitigation: Use in-place sorting or streaming input for memory efficiency.

Limited Error Handling

No validation for invalid job parameters (e.g., negative burst times).

Fix: Add assertions or exceptions for invalid inputs.

No Advanced Features

Missing real-world considerations like I/O operations, multi-core scheduling, or context-switch overhead.

Example: Round Robin could simulate context-switch delays between quantum slices.

Performance Metrics

Only individual job metrics are shown.

Improvement: Calculate averages (e.g., average waiting time) for holistic analysis.