Project Design Document For

CPU Scheduling Simulator

Mohammed ALDiri 443102724

Khaled ALBaker 443100645

Omar ALJebreen 443101949

1. Class Design and Structure

1.1 Job Class

Purpose: Represents a job, linking to its Process Control Block (PCB) for execution details.
Attributes:

- `PCB pcb`: Encapsulates job-specific data like burst time, memory requirements, and state.

Methods:

- `Job(int jobId, int burstTime, int memoryRequired)`: Initializes a job and its PCB.
- `PCB getPcb()`: Returns the associated PCB.
- `String getJobDetails()`: Logs details like ID, burst time, and memory requirements.
- `void updateJobState(State state)`: Updates the job's state using the `State` enum.
- `int compareTo(Job other)`: Compares jobs based on burst time (for SJF).

1.2 PCB (Process Control Block) Class

Purpose: Manages job attributes such as timing and state, which are essential for scheduling.
Attributes:

- **Core**: `id`, `burstTime`, `memoryRequired`, `state` (enum).
- **Timing**: `arrivalTime`, `waitingTime`, `turnaroundTime`, `remainingTime`.

Methods:

- Setters and Getters for attributes.
- `updateWaitingTime(int time)`: Adjusts waiting time.
- `updateTurnaroundTime(int time)`: Sets the turnaround time.

1.3 Scheduler Class

Purpose: Orchestrates CPU scheduling algorithms (FCFS, RR, SJF).
Attributes:

- `Queue<Job> readyQueue`: Jobs ready for execution.
- `String schedulingAlgorithm`: Algorithm in use.
- `MemoryManager memoryManager`: Manages memory operations.
- `ExecutionLog executionLog`: Logs execution details for all jobs.
- int timeQuantum: Final value having 8 as per specs.
- int Log: Static counter helps with simulations.

Methods:

- **Core**: `void run()`, `void runFCFS()`, `void runRoundRobin()`, `void runSJF()`.
- **Statistics**: `void calculateStats(String algorithm)`.
- **Logs**: `ExecutionLog getExecutionLog()`.

Design Considerations:

- Logs detailed execution events for later analysis.
- Uses separate queues for each algorithm to optimize scheduling(Scheduler).

1.4 MemoryManager Class

Purpose: Manages memory allocation, ensuring jobs transition smoothly between queues.
Attributes:

- `Queue<Job> jobQueue`: Jobs awaiting memory.
- `Queue<Job> readyQueue`: Jobs ready for execution.
- `AtomicInteger usedMemory`: Tracks current memory usage.
- `int totalMemory`: Maximum memory capacity.
- `SystemCalls systemCalls`: Handles memory operations.

Methods:

- **Memory Management**: `boolean checkMemory()`, `void allocateMemory()`, `void releaseMemory()`.
- **Job Handling**: `void run()`: Monitors queues and transitions jobs.

Design Considerations:

- Thread-safe via `AtomicInteger`.
- Clear separation of memory and job lifecycle operations(MemoryManager) (SystemCalls).

1.5 ExecutionLog Class

Purpose: Maintains detailed logs of execution events for debugging and visualization.
Attributes:

- `List<ExecutionLogEntry> logEntries`: Stores log details like start and end times.
- int startTime: logs the start time of a job.
- int endTime: logs the end time of a job.
- int remainingBurstTime: logs the remaining time of a job.
- State(enum) state: logs the state of a job.

Methods:

- `void log(int jobId, int startTime, int endTime, int remainingBurstTime, State state)`: Adds an entry.
- `List<ExecutionLogEntry> getLogEntries()`: Returns the log.
- Setters and Getters.

1.6 ReportGenerator Class

Purpose: Generates reports summarizing performance metrics and execution logs.
Methods:

- `void generatePerformanceReport(Queue<Job> jobs, String algorithmName)`: Creates performance summaries.
- `void generateExecutionLogReport(List<ExecutionLog.ExecutionLogEntry> logEntries, String algorithmName)`: Logs execution details.

Design Considerations:

- Uses Apache POI for Excel reports, ensuring professional-grade output (ReportGenerator).

1.7 State Enum

Purpose: Enumerates job lifecycle states.
Values:

- `NEW`, `READY`, `RUNNING`, `WAITING`, `TERMINATED`.

1.8 SystemCalls Class

Purpose: Provides OS-level operations to manage jobs and memory.
Methods:

- `void startProcess(Job job)`: Updates state to `RUNNING`.
- `void terminateProcess(Job job)`: Frees memory and updates state to `TERMINATED`.
- `void allocateMemory(Job job)`, `void releaseMemory(Job job)`: Handles memory operations.

---

2. Functional Workflow

1. **Job Initialization**:
    - `JobLoader` reads job data from a file, creating `Job` objects with initialized PCB details(JobLoader).
2. **Memory Allocation**:
    - `MemoryManager` checks memory availability and transitions jobs to `readyQueue` when feasible(MemoryManager).
3. **Scheduling Execution**:
    - `Scheduler` processes jobs based on the selected algorithm.
    - Logs execution details in `ExecutionLog` for debugging and analysis(Scheduler) (ExecutionLog).
4. **Performance and Reporting**:
    - `ReportGenerator` outputs performance metrics (e.g., waiting time, turnaround time) and execution logs(ReportGenerator).

---

3. Scalability and Extensibility

- **New Algorithms**: The `Scheduler` design supports easy integration of additional scheduling algorithms.
- **Thread-Safety**: Memory operations are thread-safe, ensuring consistent behavior in multithreaded environments.
- **Logging and Visualization**: Centralized execution logs facilitate debugging and data visualization.