CPU Scheduling Algorithm Visualizer

Pranay Vishwakarma
Newton School of Technology
Rishihood University

Vishesh
Newton School of Technology
Rishihood University

***Abstract*—CPU scheduling plays a crucial role in modern operating systems by managing how processes use the processor to optimize performance. However, understanding scheduling algorithms through theory alone can be challenging. To address this, we present an interactive, web-based CPU Scheduling Visualizer that simulates algorithms such as FCFS, SJF, SRTF, Round Robin, Priority Scheduling, and MLFQ. The system supports realistic scenarios including multi-core scheduling, I/O bursts, and priority aging. It features real-time Gantt charts, interactive queues, and performance metrics for instant feedback. A prediction mode further helps users test their understanding. This tool provides a practical and engaging way to learn CPU scheduling and process management.**

I. Introduction

Efficient CPU scheduling is a cornerstone of modern Operating Systems (OS), fundamentally responsible for allocating processor time to concurrent processes to maximize system utilization and minimize turnaround, waiting, and response times. While classic algorithms like First-Come, First-Served (FCFS) and Shortest Job First (SJF), alongside complex preemptive approaches like Round Robin (RR) and Multi-Level Feedback Queue (MLFQ), are critical to OS architecture, their dynamic and asynchronous nature—characterized by unpredictable arrival times, I/O bursts, and priority inversion—makes them inherently difficult to conceptualize through traditional academic instruction. Existing educational simulators often fail to bridge this conceptual gap, as they frequently restrict execution to idealized, single-core environments and omit real-world intricacies such as varied I/O wait queues and dynamic priority aging. To address these limitations, this paper presents a novel, comprehensive CPU Scheduling Visualizer: an interactive, web-based simulation framework designed to dynamically model both classic and advanced scheduling policies under realistic workloads. By tightly coupling a robust, multi-core execution engine with reactive visual analytics—including real time Gantt charts, immediate performance metrics dashboards, and a unique "Prediction Mode" for pedagogical assessment—the proposed system provides a unified, hands on environment that significantly enhances the theoretical understanding and practical evaluation of complex OS resource management dynamics.

II. System Overview

The proposed CPU Scheduling Visualizer is architected as a highly interactive and responsive web application designed to simulate, visualize, and analyze complex operating system scheduling behaviors. The system architecture is logically decoupled into three primary modules: the Core Simulation Engine, the State Management Control Plane, and the Interactive Visual Analytics UI.

A. Core Simulation Engine

At the core of the system is the simulation engine, which implements the mathematical and logical foundations of process scheduling policies. The engine supports a wide range of algorithms, including First-Come-First-Served (FCFS), Shortest Remaining Time First (SRTF), Round Robin (RR), Priority Scheduling, and the Multi-Level Feedback Queue (MLFQ).

The engine models real-world process lifecycles by supporting interleaved CPU and input/output (I/O) bursts and dynamically transitioning processes among the Ready, Running, and Waiting states. It further incorporates dynamic priority aging mechanisms and a multi-core load balancer that distributes workloads across available simulated processors.

B. State Management Control Plane

To ensure deterministic execution and responsive interface updates, the system employs a centralized immutable state management mechanism driven by a reducer function. This control plane governs the discrete tick-based simulation logic.

During each clock cycle, the control plane computes new process arrivals, handles I/O burst completions, applies priority aging thresholds, and queries the Core Simulation Engine for scheduling decisions. This modular design ensures synchronized state transitions across system entities, including ready queues and historical Gantt logs, while enabling simulation playback and control.

C. Interactive Visual Analytics UI

The user interface serves as the primary pedagogical medium, transforming internal scheduling states into immediate visual feedback. The visual analytics UI integrates multiple tightly coupled components.

1) Interactive Gantt Chart and Pipeline: A real-time timeline visualization illustrates execution sequences, context switches, and idle intervals across all CPU cores. Dynamic pipeline visualizations depict transitions between CPU execution and I/O operations.

2) Dynamic Process Tables and Kernel Logs: Centralized views display process metadata, including burst metrics and priority values, alongside real-time kernel logs that record execution events and state transitions.

Identify applicable funding agency here. If none, delete this.

3) Integrated Prediction and Evaluation Module: An embedded educational module enables users to manually compute and input expected completion time, waiting time, and turnaround time values. The system evaluates these inputs against deterministic execution results and generates multi-dimensional accuracy scorecards to reinforce theoretical understanding.

III. Algorithm Implementation and Methodology

The computational core of the visualizer is driven by a deterministic execution engine that evaluates system states at discrete time ticks. Instead of relying on static execution timelines, the engine processes dynamic state transitions and performs real-time scheduling decisions based on the selected algorithmic policy.

A. Core Data Structures and State Machine

Processes are modeled as abstract objects containing historical metadata and volatile execution states, including waiting time, turnaround time, and sequences of CPU and I/O bursts. At every clock tick, arriving processes are filtered into core-specific ready queues.

A load balancing module assigns newly arrived or returning I/O processes to the CPU core with the shortest combined ready queue and active workload.

B. Non-Preemptive and Preemptive Scheduling Logistics

Scheduling decisions are implemented through a unified routing function, `getNextProcess()`, which evaluates system states according to the selected policy.

1) Deterministic Processing (FCFS, SJF): First-Come-First-Served and non-preemptive Shortest Job First algorithms select processes only when the CPU is idle. FCFS prioritizes processes based on arrival time, while SJF sorts ready queues according to remaining CPU burst duration.

2) Preemptive Evaluation (SRTF, Preemptive Priority): For preemptive algorithms, the scheduler evaluates the ready queue at every clock tick. Context switches are triggered when a newly arrived process exhibits a shorter remaining burst time or higher scheduling priority than the currently executing process.

3) Time-Sliced Execution (Round Robin): Round Robin scheduling maintains a time quantum counter for each core. Upon quantum expiration, the active process is preempted and repositioned at the tail of the ready queue, and the quantum counter is reset.

C. Multi-Level Feedback Queue Architecture

The MLFQ implementation maintains a dynamically filtered array of priority queues mapped to discrete queue-level properties associated with each process. Time quanta increase exponentially with decreasing priority.

Processes that exhaust their assigned time quantum are demoted to lower-priority queues, while queue precedence is strictly enforced such that lower-level processes execute only when higher-priority queues are empty.

D. Starvation Mitigation Strategy (Priority Aging)

To prevent indefinite blocking under strict priority policies, the engine implements a Priority Aging mechanism. When a process exceeds a configurable waiting threshold, its numerical priority is dynamically increased.

This adaptive promotion strategy ensures bounded waiting times and guarantees eventual execution for all processes.

IV. Results and Discussion

The CPU Scheduling Visualizer was successfully deployed and evaluated as an interactive pedagogical framework. To validate the effectiveness of the proposed simulation engine, multiple predefined execution scenarios representing distinct combinations of I/O-bound and CPU-bound processes were executed across all implemented scheduling algorithms.

A. Empirical Analysis of Scheduling Algorithms

The integrated Live Metrics Dashboard enabled direct empirical comparison of algorithmic performance under identical workload conditions.

1) Interactive Metrics Observation: The system continuously monitors and aggregates critical performance metrics, including CPU utilization, system throughput, average waiting time, average turnaround time, and average response time. During experimentation, non-preemptive algorithms such as FCFS exhibited the expected convoy effect under CPU-intensive workloads, resulting in observable increases in average waiting time.

Conversely, time-sliced algorithms such as Round Robin and MLFQ maintained relatively stable response times, which were immediately visualized through real-time graphical feedback on the dashboard.

2) CPU Utilization and Multi-Core Balancing: The simulation of a multi-core execution environment demonstrated effective horizontal workload distribution through the Load Balancer module. Under single-core configurations, complex workloads frequently resulted in near-maximum CPU utilization and prolonged ready queues.

When scaled to quad-core configurations, the engine successfully partitioned the workload, leading to improved system throughput and reduced individual process turnaround times.

3) Starvation Resiliency (Priority Aging): Experiments involving strict Priority Scheduling frequently resulted in starvation of low-priority processes, reflected by increasing waiting time metrics. Upon enabling the configurable Priority Aging mechanism, the simulator demonstrated effective starvation mitigation.

Low-priority processes gradually escalated in priority and eventually obtained CPU access, thereby validating the correctness of the implemented aging model.

B. Visualization and User Interface Efficacy

Unlike conventional textbook representations, the proposed system translates abstract execution states into real-time graphical analytics, enhancing interpretability and engagement.

1) Dynamic Gantt Charts: The real-time Gantt chart proved highly effective in visualizing context-switch frequency and execution fragmentation. In SRTF and Round Robin configurations, the chart distinctly represented atomic CPU bursts and frequent preemptions, highlighting the processing overhead associated with aggressive scheduling policies.

Color-coded mappings between process identifiers and Gantt segments enabled intuitive tracking of process lifecycles across multiple CPU cores.

2) Real-Time Queue Transitions: The dynamic tabular visualization of ready queues and the universal I/O wait queue functioned as a live state machine observer. Users were able to monitor process transitions between CPU execution and I/O operations, accurately reflecting the asynchronous behavior of interrupt-driven operating systems.

C. Pedagogical Evaluation via Prediction Mode

A distinguishing feature of the proposed system is the integration of an active-learning Prediction and Evaluation module. Unlike conventional simulators that function primarily as passive visualization tools, this module enforces a predictive interaction paradigm.

Prior to simulation execution, users are required to manually estimate process completion times and average waiting time. These predictions are locked before execution, after which the deterministic engine generates actual results.

A detailed score breakdown is then produced, comparing predicted and observed values and computing objective error metrics. This feedback mechanism transforms the visualizer into an active assessment platform, reinforcing theoretical understanding through empirical validation.

V. Future Work

While the current iteration of the CPU Scheduling Visualizer successfully models complex single-node operating system mechanics, several avenues remain for future enhancement. Primary objectives for subsequent iterations include:

- **Thread-Level Concurrency:** Extending the core execution engine to differentiate between process-level and thread-level states, allowing for the simulation of user-level and kernel-level thread scheduling configurations.
- **Distributed Systems Modeling:** Adapting the load balancer and architectural model to simulate distributed resource allocation and network latency across isolated computing nodes.
- **Machine Learning Integration for Load Prediction:** Integrating historical execution data from the visualizer to train and test predictive Machine Learning algorithms capable of dynamically assigning initial Time Quanta or priority rankings for newly arriving processes based on heuristic burst profiling.
- **Exportable Analytics and Data Persistence:** Developing backend infrastructure to allow educators to save executed scenario configurations, store user prediction accuracy trends over time, and export raw timeline data for rigorous offline statistical analysis.

VI. Conclusion

This paper presented the design, architecture, and implementation of a comprehensive CPU Scheduling Visualizer aimed at bridging the conceptual gap in learning Operating System resource management. By integrating multiple layers of real-world computing intricacy—including asynchronous I/O bursts, dynamic priority aging, and multi-core load balancing—the system successfully models a wide spectrum of scheduling policies, from fundamental deterministic approaches to highly complex algorithms like the Multi-Level Feedback Queue (MLFQ).

The coupling of the mathematical simulation engine with an interactive visual UI—featuring real-time Gantt charts, live performance tracking, and dynamic queue transitions—successfully exposes the underlying micro-level execution states of the OS scheduler. Unique to this tool is the enforcement of a multi-variable Prediction assessment module, which empirically validates user comprehension across Completion Time, Waiting Time, and Turnaround Time metrics.

By providing a unified environment for interactive experimentation and formative evaluation, the CPU Scheduling Visualizer serves as a robust pedagogical instrument for the rigorous study of modern system scheduling architectures.