# CPU Scheduling Visualiser

Aviral Singh, Ashwin S, Manshu Saini
*Newton School Of Technology - RU*

***Abstract*—Understanding CPU scheduling is a fundamental aspect of operating systems education. Traditional learning methodologies often rely heavily on theoretical calculations and static diagrams, which can obscure the dynamic nature of process management. This project presents a *CPU Scheduling Visualiser*, a web-based interactive tool developed using React and Vite, backed by robust scheduling logic implemented in JavaScript. The simulator provides real-time visualization of various classical scheduling algorithms including First-Come-First-Serve (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Priority Scheduling, Round Robin (RR), and Multi-Level Feedback Queue (MLFQ). By offering dynamic Gantt charts and tracking crucial metrics such as waiting time and turnaround time, the tool bridges the gap between theoretical knowledge and practical comprehension.**

***Index Terms*—CPU Scheduling, Operating Systems, Interactive Visualization, React, Simulation, Algorithms.**

## I. Introduction

The Central Processing Unit (CPU) is one of the most vital resources in a computer system. Multiprogrammed operating systems utilize CPU scheduling algorithms to maximize CPU utilization and improve system responsiveness by seamlessly switching between processes. Despite its importance, students frequently struggle to grasp the temporal progression and decision-making logic behind preemptive and non-preemptive scheduling algorithms.

This project introduces a visualization tool engineered to demystify CPU scheduling. By providing a graphical representation of the ready queue and a dynamic Gantt chart, users can observe the runtime behavior of different algorithms under varying loads [1]. The application supports six core algorithms: FCFS, SJF, SRTF, Priority-P, Priority-NP, Round Robin, and MLFQ.

## II. Literature Review

Several educational tools have been developed over the years to assist in teaching operating systems concepts. Early simulators were largely text-based or provided rudimentary graphical interfaces that lacked interactivity. Modern web technologies have enabled the creation of rich, interactive visualizations directly in the browser without requiring native installations.

Previous works [2] highlight that visual demonstrations significantly improve knowledge retention in algorithmic concepts compared to static chalkboard examples. However, many existing tools either offer a limited set of algorithms or fail to allow flexible input parameters (like dynamic arrival times and burst times). This project builds upon these concepts by offering an intuitive, modern user interface built with React, paired with mathematically verified underlying logic modules.

## III. Methodology

### A. System Architecture

The system architecture follows a distinct separation of concerns:

- **Frontend Representation:** Developed using React and Vite, the UI manages user inputs (process properties, algorithm selection) and renders resulting metrics and Gantt charts dynamically.
- **Algorithm Logic:** The logic is encapsulated in isolated ES6 modules (e.g., `FCFS.js`, `SJF.js`). Each module implements a uniform interface containing methods like `selectNext()` and `shouldPreempt()`.

### B. Implemented Algorithms

The simulator evaluates six baseline scheduling approaches:

1) **First-Come-First-Serve (FCFS):** A straightforward, non-preemptive algorithm that processes jobs strictly by chronological arrival.
2) **Shortest Job First (SJF):** Selects the process with the smallest execution burst time, mathematically proven to minimize average waiting time but susceptible to starvation.
3) **Shortest Remaining Time First (SRTF):** The preemptive counterpart to SJF. The scheduler dynamically re-evaluates the queue whenever a new process arrives.
4) **Priority Scheduling:** Allocates CPU based on an assigned priority integer.
5) **Round Robin (RR):** A preemptive, time-sharing algorithm utilizing a predefined time quantum ($q$) to cycle through processes fairly.
6) **Multi-Level Feedback Queue (MLFQ):** A complex scheduling structure where processes are dynamically moved between queues of varying priorities based on their CPU usage behavior, balancing response time and throughput.

### C. Evaluation Metrics

For each simulated execution, the system calculates and displays the following core metrics for every process $i$:

- **Completion Time ($CT_i$):** The exact time the process finishes execution.
- **Turnaround Time ($TAT_i$):** The total time taken from arrival to completion.

$$TAT_i = CT_i - ArrivalTime_i \tag{1}$$

- **Waiting Time ($WT_i$):** The time a process spends waiting in the ready queue.

$$WT_i = TAT_i - BurstTime_i \tag{2}$$

## IV. Results and Observations

The application successfully models edge-cases inherent to CPU scheduling, such as the 'Convoy Effect' in FCFS where short processes are bottlenecked behind a CPU-intensive process. When visualizing Shortest Remaining Time First (SRTF), the dynamic preemptions correctly slice execution blocks on the timeline, vividly illustrating context switching.

TABLE I
Theoretical Comparison of Simulated Algorithms

| Algorithm | Preemptive | Fairness | Primary Drawback |
|---|---|---|---|
| FCFS | No | Poor | Convoy Effect |
| SJF | No | Poor | Needs knowledge of future burst times |
| SRTF | Yes | Poor | Starvation of long jobs |
| Priority | Both | Poor | Starvation (requires aging) |
| Round Robin | Yes | High | High context-switch overhead |
| MLFQ | Yes | Fair | Complex configuration of queues and quantums |

The interactive UI allows users to easily recognize these characteristics (as summarized in Table I) by manipulating process parameters in real-time.

## V. Conclusion and Future Work

The CPU Scheduling Visualiser accomplishes its goal of providing an accessible, immediate, and accurate representation of OS process scheduling. By combining modern web development practices with strict computer science principles, it serves as an excellent reference tool.

Future enhancements for this project could include:

- Directly compiling the C core logic to WebAssembly (Wasm) to handle massive datasets inside the browser with native performance.
- Visualizing context-switch overhead and memory utilization parameters in the timeline.

## References

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Wiley, 2018.

[2] J. M. Stewart, "Teaching Operating Systems: Using Simulation and Visualization," *Journal of Computing Sciences in Colleges*, vol. 23, no. 4, pp. 110–115, 2008.

[3] W. Stallings, *Operating Systems: Internals and Design Principles*, 9th ed. Pearson, 2017.