An Advanced Interactive Web-Based CPU Scheduling Visualizer Supporting Classical, Multi-Level, Proportional-Share, and I/O-Aware Algorithms

Divyanshi Sachan (230069)
Department of Computer Science and Engineering
Rishihood University, India

Subham Mahapatra (230037)
Department of Computer Science and Engineering
Rishihood University, India

***Abstract*—This paper presents a comprehensive web-based CPU Scheduling Visualizer implementing fifteen scheduling algorithms across classical, fairness-based, multilevel, proportional-share, and I/O-aware paradigms. The system enables real-time simulation, animated Gantt chart visualization, smart algorithm switching, and multi-algorithm comparison. Experimental evaluation demonstrates algorithmic behavior differences in terms of waiting time, turnaround time, context switching, and CPU utilization.**

***Index Terms*—CPU Scheduling, MLFQ, HRRN, Lottery Scheduling, Stride Scheduling, Operating Systems Education**

I. Introduction

CPU scheduling determines which process is allocated the CPU at any given time. It directly influences system performance metrics including waiting time, turnaround time, throughput, response time, and CPU utilization.

Traditional approaches rely on static examples. However, modern scheduling policies involve preemption, priority migration, proportional fairness, and I/O blocking. To address this complexity, we developed an interactive simulator supporting fifteen scheduling algorithms with real-time visualization and experimental comparison.

II. Features of the Simulator

The developed simulator provides:

- Real-time Gantt chart visualization
- Step-by-step playback with ready queue animation
- Support for 15 scheduling algorithms
- Multi-algorithm comparison (up to four)
- Smart heuristic-based algorithm switching
- I/O-aware scheduling simulation
- Proportional-share scheduling (Lottery, Stride)
- Custom user-defined scheduling logic
- Export options (CSV, JSON, PNG)
- URL-based state sharing

III. Formal Model

Let:

$$P = \{P_1, P_2, ..., P_n\}$$

Each process:

$$P_i = (AT_i, BT_i, PR_i)$$

Selection rule:

$$P_k = \arg\max f(P_i, t)$$

IV. Scheduling Algorithms

Each algorithm includes explanation, formula, and complexity.

A. 1. FCFS

Definition: Executes processes in arrival order.

$$P_k = \arg\min(AT_i)$$

Time Complexity: $O(n \log n)$
Limitation: Convoy effect.

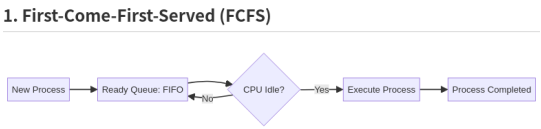

Fig. 1. FCFS Scheduling

B. 2. SJF (Non-Preemptive)

$$P_k = \arg\min(BT_i)$$

Minimizes average waiting time.
Complexity: $O(n \log n)$

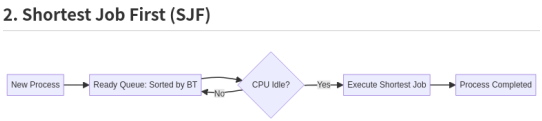

Fig. 2. SJF Scheduling

C. 3. SRTF

$$P_k = \arg\min(Remaining_i)$$

Preemptive SJF.
Complexity: $O(n \log n)$

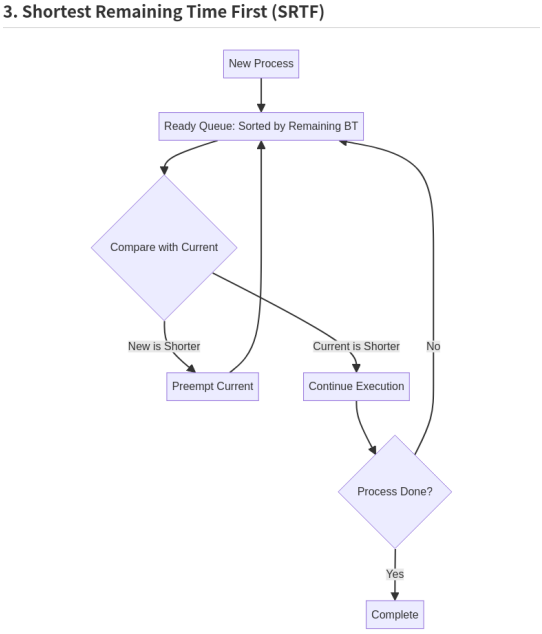

Fig. 3. SRTF Scheduling

D. 4. Round Robin

Each process receives time quantum q.

$$Remaining_i = Remaining_i - q$$

Complexity: $O(n)$ per cycle.

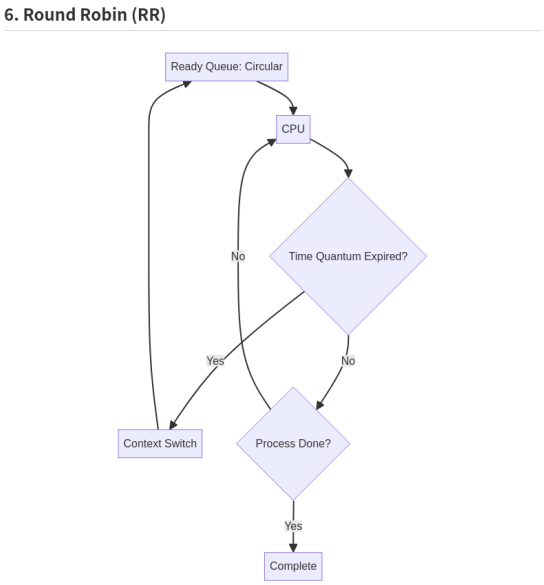

Fig. 4. Round Robin Scheduling

E. 5. HRRN

$$RR_i = \frac{WT_i + BT_i}{BT_i}$$

$$P_k = \arg\max(RR_i)$$

Complexity: $O(n^2)$

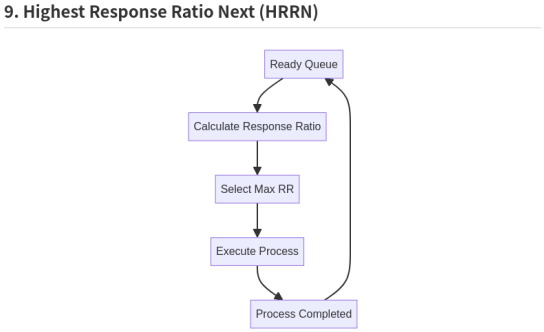

Fig. 5. HRRN Scheduling

F. 6. MLFQ

Dynamic feedback queues.

$$Queue_i = Queue_i + 1$$

Complexity: $O(n \log n)$

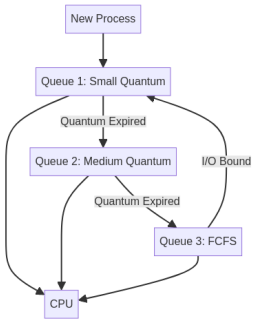

Fig. 6. MLFQ Scheduling

G. 7. Lottery Scheduling

$$P(P_i) = \frac{T_i}{\sum T_j}$$

Random proportional fairness.
Complexity: $O(n)$

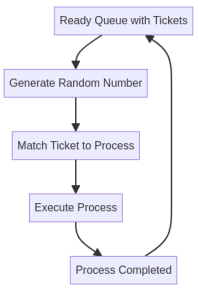

Fig. 7. Lottery Scheduling

H. 8. Stride Scheduling

$$Stride_i = \frac{L}{T_i}$$

$$Pass_i = Pass_i + Stride_i$$

$$P_k = \arg\min(Pass_i)$$

Complexity: $O(\log n)$ using heap.

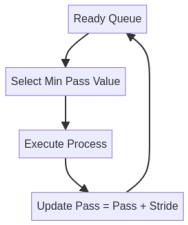

Fig. 8. Stride Scheduling

V. Performance Metrics

$$TAT = CT - AT$$

$$WT = TAT - BT$$

$$Throughput = \frac{n}{\max(CT)}$$

$$CPUUtilization = \frac{BusyTime}{TotalTime} \times 100$$

VI. Experimental Results

A. FCFS Execution

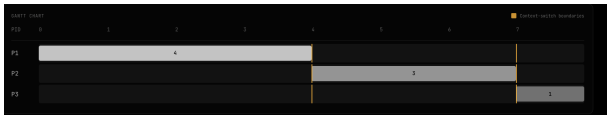

Fig. 9. Gantt Chart Output for FCFS

B. Per-Process Metrics

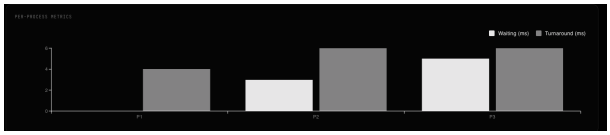

Fig. 10. Per-Process Waiting and Turnaround Time

C. Algorithm Comparison

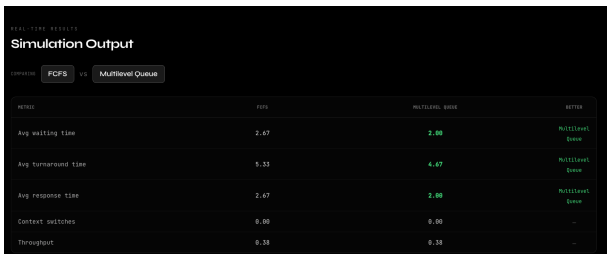

Fig. 11. Metric Comparison Table

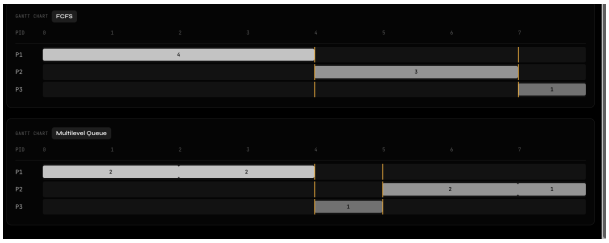

Fig. 12. Gantt Chart Comparison between FCFS and MLQ

Results demonstrate significant differences in waiting time and context switching across algorithms.

VII. Conclusion

This work presents a comprehensive interactive CPU Scheduling Visualizer integrating fifteen algorithms across multiple scheduling paradigms. The system enhances theoretical understanding through visualization and experimental analysis.

References

[1] A. Silberschatz et al., *Operating System Concepts*, 10th ed., Wiley, 2018.
[2] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed., Pearson, 2015.