Disk Scheduling Algorithm Simulator

Operating Systems Project Report

Gangadhar Yadav, Samyak Mittal, Ayush Dev
Rishihood University, Delhi NCR, India
Course: Operating Systems
February 2026

***Abstract*—This project implements a Disk Scheduling Algorithm Simulator to demonstrate how an operating system schedules disk I/O requests to minimize seek time and improve performance. The simulator includes eight algorithms—FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK, N-Step SCAN, and FSCAN—with a web-based interface for simulation and comparison. The backend is developed using Django and Django REST Framework, while the frontend is implemented in React with graphical visualization using Recharts. The system computes total seek time, average seek time, and performance metrics to compare algorithms and identify the optimal scheduling strategy for a given input.**

***Index Terms*—Disk scheduling, FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK, N-Step SCAN, FSCAN, Seek time, Operating Systems, I/O management.**

I. Introduction

In modern operating systems, multiple processes may request disk access simultaneously. Efficient disk scheduling is essential to reduce seek time and improve overall system throughput. Disk scheduling algorithms determine the order in which disk I/O requests are serviced.

The primary objective of this project is to design and implement a simulator that demonstrates the behavior of different disk scheduling algorithms and compares their performance. The simulator provides visualization and metrics to better understand trade-offs between fairness and performance.

II. Problem Statement and Objectives

A. Problem Statement

Design and implement a simulator that:

- Accepts a list of track requests and initial head position.
- Simulates eight disk scheduling algorithms.
- Computes total and average seek time.
- Compares algorithms and highlights the best-performing one.

B. Objectives

- Implement FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK, N-Step SCAN, and FSCAN.
- Build REST APIs for simulation and comparison.
- Provide interactive visualizations.
- Analyze fairness and starvation issues.

III. System Architecture

A. Overall Design

The system follows a two-tier architecture:

- **Backend:** Django + Django REST Framework (Python)
- **Frontend:** React 18 + Recharts

The frontend communicates with backend REST APIs using JSON.

B. API Endpoints

- `GET /api/`
- `GET /api/algorithms/`
- `POST /api/simulate/`
- `POST /api/compare/`

IV. Algorithms Implemented

TABLE I
Disk Scheduling Algorithms

Algorithm	**Description**
FCFS	First Come First Served
SSTF	Shortest Seek Time First
SCAN	Elevator algorithm
C-SCAN	Circular SCAN
LOOK	SCAN without reaching disk ends
C-LOOK	Circular LOOK
N-Step SCAN	Batched SCAN processing
FSCAN	Dual-queue SCAN

V. Implementation Details

A. Backend

The backend consists of:

- Algorithm module implementing scheduling logic.
- Request validation layer.
- Metrics computation module.
- REST API controllers.

Input parameters:

- Requests (track list)
- Initial head position
- Disk size
- Direction (for SCAN variants)
- N-step batch size

B. Frontend

Main components:

- Input Panel
- Visualization Panel
- Results Panel
- Comparison Panel

The system visualizes:

- Head movement graph
- Seek distance
- Bar chart comparison

VI. Performance Metrics

The simulator computes:

- Total Seek Time
- Average Seek Time
- Throughput
- Efficiency

The best algorithm is identified based on minimum total seek time.

VII. Results and Discussion

A. Sample Input

- Requests: 98, 183, 37, 122, 14, 124, 65, 67
- Initial Position: 53
- Disk Size: 200 tracks

B. Observations

- FCFS ensures fairness but may increase total seek time.
- SSTF reduces seek time but may cause starvation.
- SCAN-family algorithms balance fairness and efficiency.
- LOOK variants reduce unnecessary disk traversal.
- N-Step SCAN and FSCAN reduce starvation via batching.

VIII. Conclusion

The Disk Scheduling Algorithm Simulator successfully demonstrates eight disk scheduling techniques and highlights their trade-offs in fairness, starvation, and performance. The web-based visualization improves conceptual understanding of disk head movement and scheduling behavior. Future enhancements may include animated simulations, additional algorithms, and exportable reports.

Acknowledgment

We thank the Operating Systems faculty at Rishihood University for guidance and support throughout this project.

References

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

[2] Django REST Framework. [Online]. Available: https://www.django-rest-framework.org/

[3] React Documentation. [Online]. Available: https://react.dev/

[4] Recharts Documentation. [Online]. Available: https://recharts.org/