A

Project Report

On

File System Simulator

Submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Rakshita Joshi 2261462

Abhishek Majila 2261066

Vivek Melkani 2261620

Pankaj Singh Raikwal 2261410

Under the Guidance of

Mr.Prince Kumar

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI,

DISTRICT- NAINITAL-263132

2024-2025

STUDENT'S DECLARATION

We, **Rakshita Joshi, Abhishek Majila, Vivek Melkani,Pankaj Singh Raikwal** hereby declare the work, which is being presented in the project, entitled **' File System Simulator'** in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of Mr. Prince Kumar.

The matter embodied in this project has been submitted by us for the award of B.Tech degree.

Date:

Rakshita Joshi

Abhishek Majila

Vivek Melkani

Pankaj Singh Raikwal

CERTIFICATE

The project report entitled **"File System Simulator"** being submitted by Rakshita Joshi 2261462, Abhishek Majila 2261066, Vivek Melkani 2261620, Pankaj Singh Raikwal 2261410 of B.Tech.(CSE) to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

Mr.Prince Kumar
(Assistant Professor)

Dr. Ankur Singh Bisht
(HOD, CSE)

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director **'Prof. (Col.) Anil Nair (Retd.)',** GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president **'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.Many thanks to **'Dr. Ankur Singh Bisht'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide **'Prince Kumar'** (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

Rakshita Joshi 2261462
Abhishek Majila 2261066
Vivek Melkani 2261620
Pankaj Singh Raikwal 2261410

Abstract

The File System Simulator project simulates core functionalities of an operating system's file system using Python. It is designed to provide a hands-on educational tool for understanding how file systems manage data, allocate memory, and perform scheduling tasks. Key modules of the project emulate operations such as file creation, deletion, and directory management. Additionally, the project incorporates storage allocation strategies and disk scheduling algorithms to model how data is read and written efficiently. The modular structure supports extensibility and offers a clear demonstration of concepts like file hierarchy, block allocation, and seek-time optimization. This simulation aids learners in visualizing and interacting with file system mechanics, bridging the gap between theory and practical implementation.

The project focuses on core Operating System (OS) concepts, including:

- **File and Directory Management:** Users can create, delete, and navigate directories and files, mimicking the hierarchical file system of modern operating systems.
- **Memory and Storage Allocation**: Implements various allocation strategies such as contiguous, linked, and indexed allocation to demonstrate how files occupy storage blocks.
- **Disk Scheduling Algorithms**: Includes simulations of popular algorithms like FCFS (First-Come-First-Serve), SSTF (Shortest Seek Time First), and SCAN to model the optimization of disk I/O operations.
- **Data Persistence**: Utilizes JSON to simulate storage tracking and metadata management, reflecting how real systems maintain file information.

TABLE OF CONTENTS

CHAPTER-1

INTRODUCTION

1.1 Prologue

The File System Simulation Project is a Python-based educational tool that models how an operating system manages files, directories, and disk operations. Built with a Streamlit interface, it provides a real-time, interactive platform to explore core OS concepts like file management, storage allocation, and disk scheduling. This project aims to simplify the complex behaviors of file systems into visual, understandable operations.

1.2 Background and Motivation

Understanding file systems is a core part of operating systems, yet students often struggle with its abstract nature. Traditional teaching methods lack hands-on experience, making it difficult to grasp file allocation, directory structures, and disk scheduling. Motivated by this gap, I chose to build an interactive simulation tool using Python and Streamlit. This project allows users to visualize and interact with core OS concepts in real time, making learning more engaging. My goal was to simplify learning through simulation, enhance conceptual clarity, and contribute a meaningful educational resource for students studying systems programming.

1.3 Problem Statement

Most educational platforms and textbooks present operating system concepts—particularly file systems—in a purely theoretical manner, making them difficult for students to understand and apply. File operations, directory structures, storage allocation techniques, and disk scheduling algorithms are often taught without hands-on tools or visual representations. This lack of interactivity creates a gap between theoretical learning and practical comprehension. Students

struggle to visualize how files are stored, how memory is allocated, or how the disk scheduling works in real scenarios. Therefore, there is a strong need for a simple, intuitive, and interactive tool that simulates these processes in a visual manner. Such a platform would significantly enhance the learning experience and help learners build a deeper understanding of core OS functionalities through active exploration and real-time feedback.

1.4 Objectives and Research Methodology

The main objective of this project is to create an interactive, educational tool that simulates the core functions of a file system in an operating system. It aims to make OS concepts like file management, memory allocation, and disk scheduling more accessible and engaging through a user-friendly graphical interface. By using Python and Streamlit, the project combines simplicity in coding with powerful visualization capabilities. The research methodology involved studying real OS mechanisms, analyzing different file allocation and scheduling algorithms, and translating them into modular Python components for simulation.

Key Objectives:

- Simulate file system operations such as create, delete, and navigate.
- Implement storage allocation methods (contiguous, linked, indexed).
- Visualize disk scheduling algorithms (FCFS, SSTF, SCAN).
- Provide a responsive GUI using Streamlit for user interaction.
- Use JSON for lightweight data persistence and state management.
- Offer a modular backend structure for clarity, flexibility, and future scalability.

1.5 Project Organization

The project was completed through collaborative teamwork with clear role distribution to ensure efficient development and learning:

1. Frontend – Built with Streamlit

- The project uses Streamlit to create a simple and interactive web interface.
- Streamlit allows users to run the program in their browser and perform tasks like creating files, managing directories, and running disk scheduling algorithms.
- It includes buttons, forms, and charts to make the experience user-friendly and responsive.

2. Backend – Python Modules

- The backend is written in Python and organized into different modules.
- Each module handles a specific task:
    - **Directory management** – for creating and deleting folders.
    - **File operations** – for creating, reading, and deleting files.
    - **Storage allocation** – shows how files are stored on disk (contiguous, linked, indexed).
    - **Disk scheduling** – runs algorithms like FCFS, SSTF, and SCAN to manage disk access.
- This setup makes the code neat, easy to understand, and works like a basic operating system.

3. Visualization – Real-Time Display

- The project shows real-time visuals using Streamlit's features.
- Users can see:
    - How files are stored in disk blocks.
    - Charts showing the movement of the disk head using different scheduling algorithms.
    - The structure of folders and files.

- These visuals help users better understand how a file system works.

4. Optional Database – JSON for Saving Data

- The project uses a JSON file instead of a full database.
- It saves information like file names, folders, and disk usage.
- This makes the project lightweight and easy to run, while still showing how data is stored and managed.

CHAPTER-2

HARDWARE AND SOFTWARE REQUIREMENTS

2.1Hardware Requirement

Component	Minimum Requirement	Recommended
Processor	Intel Core i3 / AMD equivalent	Intel Core i5 or higher
RAM	4 GB	8 GB or more
Storage	100 MB free disk space	500 MB free disk space
Display	1024×768 resolution	1366×768 or higher
Input Device	Standard Keyboard & Mouse/Touchpad	Same
Internet	Optional (for installing packages)	Stable internet for updates

2.2 Software Requirement

For Windows-

Component	Minimum Requirement
Operating System	Windows 7 / 8 / 10 / 11
Python Version	Python 3.7
Libraries Required	streamlit, os, json, math, time
IDE / Editor	Notepad / CMD / IDLE
Browser	Any modern browser (Chrome, Firefox)

For Linux-

Component	Minimum Requirement
Operating System	Ubuntu 18.04
Python Version	Python 3.7
Libraries Required	streamlit, os, json, math, time
IDE / Editor	Nano / Vim / Geany
Browser	Chrome / Firefox
Package Manager	APT, YUM, or Pacman

For MacOs-

Component	Minimum Requirement
Operating System	macOS (10.15)
Python Version	Python 3.7
Libraries Required	streamlit, os, json, math, time
IDE / Editor	IDLE / Terminal
Browser	Safari / Chrome
Package Manager	Homebrew

Chapter-3

Coding Of Functions

This project uses Python and Streamlit to simulate core concepts of an Operating System, such as file management, memory allocation, and disk scheduling. The code is organized modularly, making each component responsible for a specific part of the simulation. Below is a breakdown of the main functional components:

1. File and Directory Management

These functions simulate creating, deleting, and navigating files and directories.

Key Functions:

- create_file(name, size)
- delete_file(name)
- create_directory(name)
- navigate_directory(path)

Description:

- The create_file() function takes a file name and size and creates a new file in the current directory. It also checks whether enough space is available.
- delete_file() removes a file by clearing its allocation and updating the directory tree.
- create_directory() allows the user to build a directory hierarchy similar to real OS behavior.
- navigate_directory() lets users simulate moving through folders like a file explorer.

These functions use Python dictionaries and lists to represent directory trees and file metadata. They update both the visual output and the internal JSON state.

2. Storage Allocation Techniques

The project includes simulation of three classic memory allocation strategies: Contiguous**,** Linked**,** and Indexed Allocation**.**

Key Functions:

- allocate_contiguous(file_name, file_size)
- allocate_linked(file_name, file_size)
- allocate_indexed(file_name, file_size)

Description:

- allocate_contiguous() checks for a continuous block of free memory. If found, it marks those blocks as used and assigns them to the file.
- allocate_linked() allocates random free blocks and links them like a chain, storing references in a linked structure.
- allocate_indexed() creates an index block that contains all the addresses of the actual data blocks.

3. Disk Scheduling Algorithms

To demonstrate how disk head movement is optimized, the following algorithms are implemented:

Key Functions:

- fcfs(requests)
- sstf(requests, head)
- scan(requests, head, direction)

Description:

- fcfs() (First-Come First-Serve) processes disk I/O requests in the order they arrive.

- sstf() (Shortest Seek Time First) chooses the request nearest to the current head position, reducing head movement.
- scan() moves the head in one direction, servicing all requests along the path, then reverses direction (elevator algorithm).

4. Streamlit GUI Integration

The file app.py is the main entry point and integrates all backend functions with an interactive web interface using Streamlit**.**

Key Features:

- Sidebar options to choose algorithms and input file details.
- Buttons for creating/deleting files and directories.
- Visual block diagram to show memory allocation.
- Charts for disk scheduling visualization.

Description:

Streamlit widgets like st.button(), st.selectbox(), and st.text_input() are used to capture user input. Based on this input, the backend functions are called and the output is displayed immediately, making the experience dynamic and educational.

5. Data Persistence and Utilities

The system state (directory structure and disk usage) is saved using JSON files.

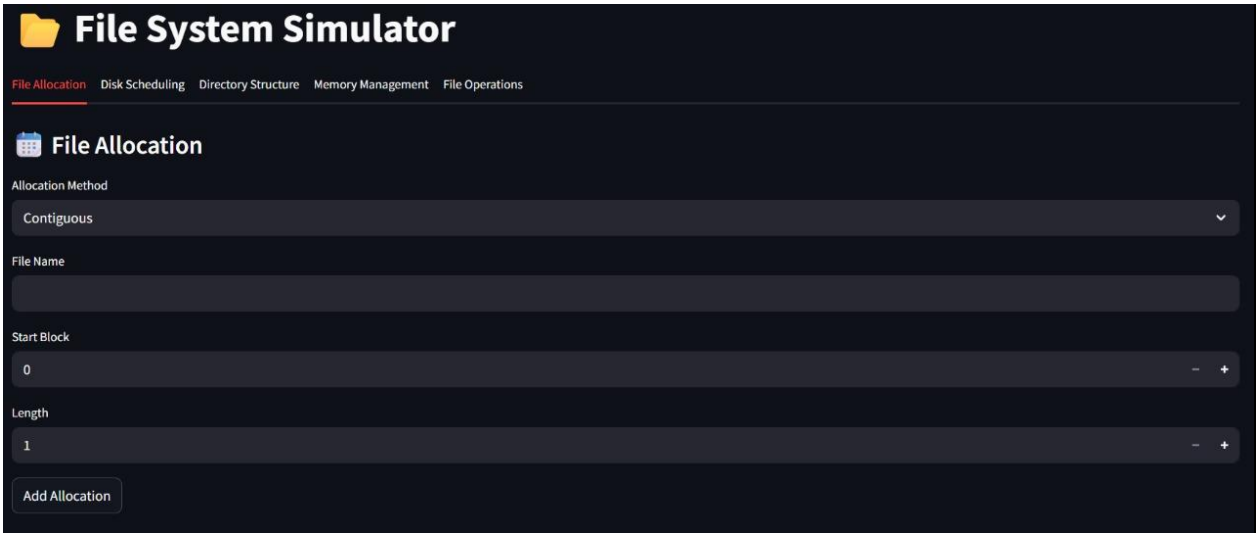
Functions:

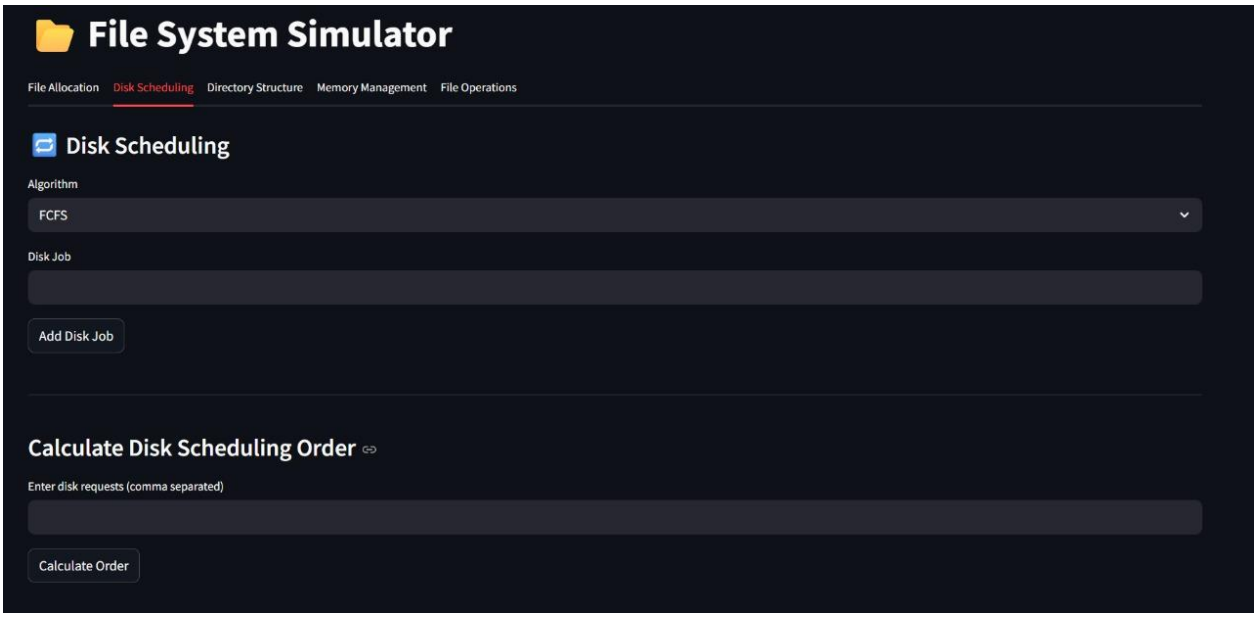
- save_state_to_json()
- load_state_from_json()

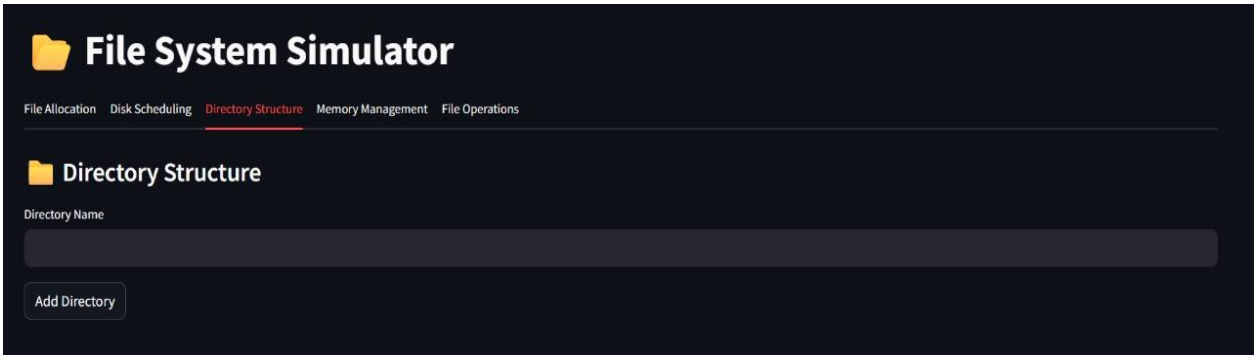
These ensure that the simulation retains its state between sessions or after page reloads.

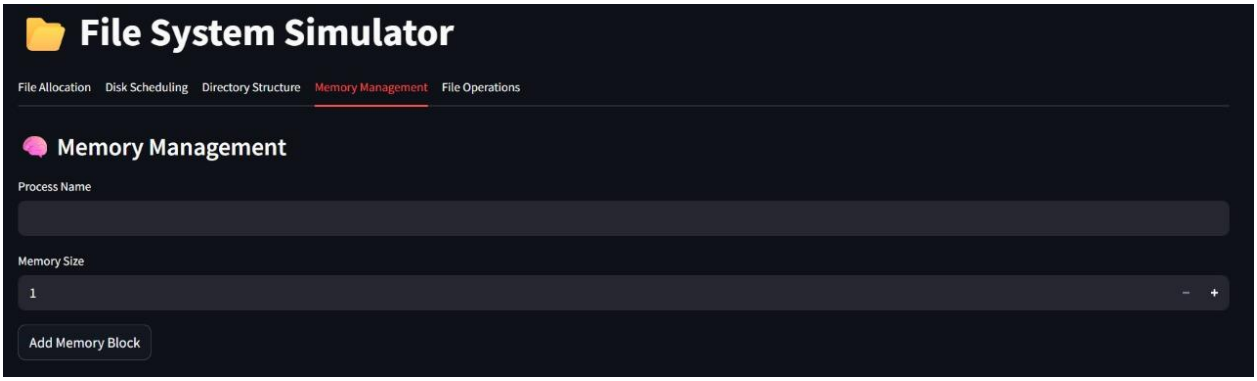
Chapter-4

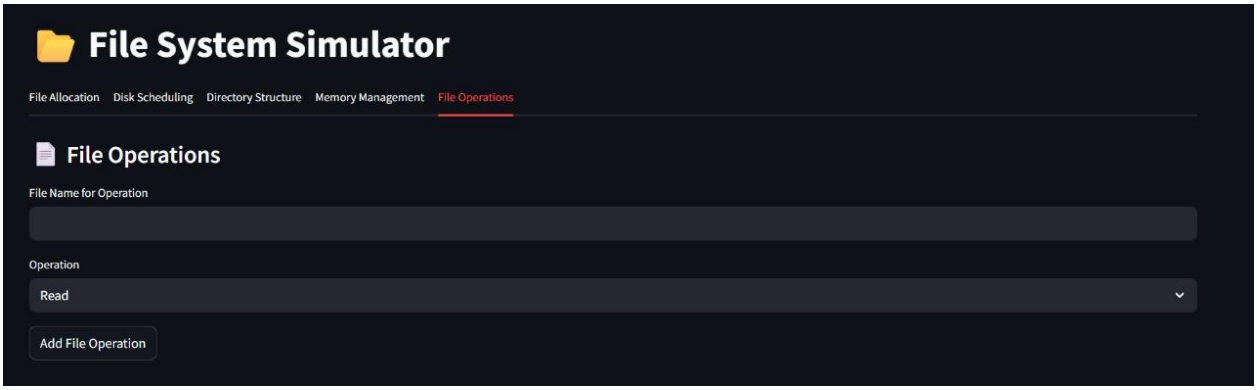
SNAPSHOTS






File System Simulator
File Allocation
Disk Scheduling
Directory Structure
Memory Management
File Operations
Directory Structure
Directory Name
Add Directory


File System Simulator
File Allocation
Disk Scheduling
Directory Structure
Memory Management
File Operations
Memory Management
Process Name
Memory Size
1
Add Memory Block


File System Simulator
File Allocation
Disk Scheduling
Directory Structure
Memory Management
File Operations
File Operations
File Name for Operation
Operation
Read
Add File Operation

Chapter-5

LIMITATIONS

While the File System Simulator provides a highly visual and interactive learning tool for understanding operating system concepts, it also has a few limitations that are important to acknowledge.

1. Limited Real-World Integration

The simulator simplifies many concepts for learning purposes, but:

- It does not interact with actual file systems on the host machine.
- Real-world OS behaviors such as caching, concurrency, or access permissions are not covered.

This makes the tool educational but not suitable for real-time system-level operations.

2. Simplified Allocation and Scheduling Models

- Only basic allocation methods (like contiguous, linked, or indexed) are simulated.
- Disk scheduling algorithms are idealized and may not reflect hardware-specific optimizations.

This simplification helps users understand the logic but may limit understanding of more advanced real-world scenarios.

3. Scalability Constraints

- The simulator is built for small-scale demos.
- Handling large file systems or thousands of operations may slow down performance or result in UI lag.

4. Lack of Persistent Storage

- The simulator does not use a database to save user inputs or results.

- All data is lost upon refreshing or closing the application.

Adding persistent storage would improve its usefulness for long sessions or record-keeping.

5. **No Multi-user Support**

- It is a single-user desktop web app.
- Features like user authentication, file sharing, or collaborative usage are not included.

The File System Simulator effectively visualizes OS concepts but has several limitation.

It lacks integration with real operating systems and simplifies file allocation and disk scheduling models. The tool is not scalable for large systems and doesn't support persistent storage—data is lost on refresh. Additionally, it offers no multi-user capabilities or advanced OS functionalities like concurrency or permissions. Despite these constraints, it serves well as an educational tool for learning basic file system operations and memory management.

Chapter-6

ENHANCEMENTS

While the current implementation of the File System Simulator successfully visualizes essential file system and OS concepts, there are several areas where the project can be significantly improved to offer a richer, more realistic, and more educational experience.

1. Persistent Data Storage

To enhance usability:

- Integrate a lightweight database such as SQLite or a NoSQL solution like MongoDB.
- Enable the saving of file allocation tables, directory structures, and scheduling logs.

This would allow users to resume sessions, review historical data, and support longer and more complex simulations.

2. User Authentication and Profiles

Introducing user login and profile features would personalize the experience:

- Users could save their simulation progress and configurations.
- Different roles (e.g., student, instructor) could be added to support classroom use.

3. Real-Time Visualization Enhancements

Improved visual feedback can make learning more intuitive:

- Use color-coded blocks and animated diagrams to represent memory or disk scheduling operations.
- Integrate Streamlit-compatible libraries like Plotly or Altair for interactive charts.

4. More File Allocation and Scheduling Algorithms

Expand the list of implemented algorithms:

- Add support for more complex allocation methods such as multilevel indexing and extent-based allocation.
- Include additional disk scheduling algorithms like LOOK, C-LOOK, and multi-level queue scheduling.

This would provide a broader range of learning scenarios for advanced users.

5. Error Handling and Validations

Currently, the simulator might not handle invalid inputs robustly:

- Add input validation for block sizes, file names, and length values.
- Provide meaningful error messages and suggestions.

This would help users avoid confusion and better understand proper configurations.

6. Cross-Platform Packaging

Package the application for offline use:

- Create installable desktop versions using tools like PyInstaller or Electron.
- Provide Linux, Windows, and macOS compatibility for broader access.

7. Interactive Tutorials and Hints

To make the tool even more educational:

- Include guided walk-throughs for each OS concept.
- Add tooltips and documentation within the UI for better understanding.

Chapter-7

CONCLUSION

The File System Simulator project serves as an effective educational tool designed to bridge gap between theoretical concepts of operating systems and their practical understanding. Built using Streamlit, this application provides an interactive, user-friendly interface for simulating essential OS mechanisms such as file allocation strategies, disk scheduling algorithms, directory structures, and basic memory management techniques. Through clean graphical interface we can visualize how files are stored, accessed, and managed within a system, enhancing their comprehension through hands-on interaction rather than abstract explanation.

The modular backend structure enables each component—whether file allocation or disk scheduling - to operate independently while still contributing to a cohesive learning experience. This simulation-focused approach not only demystifies complex algorithms but also fosters deeper engagement, especially for learners with limited exposure to low-level programming or system design. While the tool has limitations such as lack of persistent storage, simplified logic, and absence of real-time OS interaction, it still provides an excellent foundation for academic use. These constraints also present opportunities for future development and enhancement, making the project a scalable and extendable learning platform.

The File System Simulator project successfully meets its objective of making operating system concepts more accessible and engaging. It proves that with right tools and interfaces, even complex system-level operations can be understood in an intuitive and enjoyable way. This project is ideal for students, educators, and self-learners who wish to build a strong conceptual foundation in file systems and related OS topics. With further improvements, it has potential to become complete e-learning module for operating system education.

REFERENCES

1) Robbins, Stephen P., & Judge, Timothy A. (2007). Organizational Behaviour (12th ed.). New Delhi: Prentice-Hall of India.
2) Yammarino, Francis J., Spangler, William D., & Dubinsky, A. J. (1998). Transformational and Contingent Reward Leadership: Individual, Dyad and Group Level of Analysis. Leadership Quarterly, 9, 27–54.
3) Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.
4) Stallings, W. (2017). Operating Systems: Internals and Design Principles (9th ed.). Pearson Education.
5) Tanenbaum, A. S., & Bos, H. (2015). Modern Operating Systems (4th ed.). Pearson Education.