Project Title: File System Navigation

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 🡪 Table of Contents

1. Introduction

2. Team Members

3. Objective

4. Problem Statement

5. System Requirements

6. Data Structures Used

7. Module Architecture and Descriptions

8. Algorithmic Approaches

9. Sample input and output

10.Future Scope and Enhancements

11. Conclusion

12. References

# 1. Introduction

🡺The File System Navigation project is designed to simulate the core functionalities of a hierarchical file system within a command-line interface. The system allows users to perform essential directory management operations such as creation, deletion, and traversal, employing commands analogous to those in conventional operating system shells. This simulation aims to deepen understanding of the underlying data structures and command parsing techniques involved in file system management.

# 2. Team Members

|  |  |  |  |
| --- | --- | --- | --- |
| S.No. | Name | Roll Number | Responsibility |
| 1. | Pavan naidu .S | 69 | Team lead ,project developed , File I/O and data Management,  Testing, validation,  and report. |
|  |  |  |  |
|  |  |  |  |

# 3. Objective

To develop an intuitive and interactive command-line application that:

* Accurately replicates file system navigation operations.
* Provides users with a practical environment to explore directory structure manipulation.
* Demonstrates underlying algorithmic implementations.

# 4. Problem Statement

🡺While modern operating systems provide comprehensive file management capabilities, their complexity can be a barrier to learners. This project addresses the challenge by building a simplified file system navigator that encapsulates core functionalities, enabling users to grasp fundamental data structure applications and algorithmic problem-solving techniques inherent in directory traversal and management.

# 5. System Requirements

1. Hardware Requirements

* **Processor :** Intel i5 or higher (or equivalent)
* **RAM :**Minimum 2 GB (4 GB recommended)
* **Storage :** At least 100 MB free disk space
* **Input Devices:** Keyboard (for command-line interaction)
* **Display :** Standard console/terminal window

2. Software Requirements

* **Operating System:** Windows / Linux / macOS
* **Compiler/Interpreter:** GCC for C / Python interpreter (if implemented in Python)
* **Development Environment:** Any text editor or IDE (e.g., VS Code, Code::Blocks, Turbo C for C)
* **Command-line Interface:** Native terminal or command prompt

# 6. Data Structures Used

🡪Tree & Structure:

Utilized to represent the hierarchical organization of directories, where each node corresponds to a directory that may have multiple child directories.

🡪Stack:

Employed to track the navigation history, particularly useful for implementing the cd .. (move to parent directory) operation.

🡪Queue/LinkedList(Optional):

May be used to maintain command history or for thread-safe operations in advanced implementations

# 7. Module Architecture and Descriptions

|  |  |
| --- | --- |
| **Module** | **Description** |
| Main Module | Entry point of the application, handles program initialization, input/output flow, and coordinates with other modules. |
| File System Module | Manages the directory tree structure. Supports operations such as creating, deleting, renaming, and traversing directories/files. |
| Command Parser | Reads, parses, and validates user input. Maps commands like cd, ls, mkdir, rmdir to corresponding file system operations. |
| Navigation Module | Provides navigation functions (move to parent, child, or specific path) and maintains the current working directory. |
| Logger (Optional) | Stores logs of user commands and system responses for debugging, auditing, and testing purposes. |
| Error Handler | Handles invalid commands, incorrect paths, and exceptions, ensuring smooth user experience. |
| Helper/Utility Module | Provides helper functions (e.g., formatting output, string handling, path resolution) used across different modules. |

8. Algorithms Approaches

### **1. Linked List Operations**

**a) Insert (linkedListInsert)**

Step 1: Create a new node with the given name.

Step 2: If the head is NULL, set head = new node.

Step 3: Else, traverse the list until the last node.

Step 4: Link the last node’s next pointer to the new node.

Step 5: End.

**b) Delete (linkedListDelete)**

Step 1: Initialize current = head, prev = NULL.

Step 2: Traverse the list until current is NULL.

Step 3: If current->name matches:

- If prev == NULL, move head = current->next.

- Else, set prev->next = current->next.

- Free current and return success.

Step 4: Continue until end. If not found, return failure.

**c) Search (linkedListContains)**

Step 1: Start from head.

Step 2: Compare each node’s name with the given name.

Step 3: If found, return true.

Step 4: If end of list reached, return false.

**d) Traverse (linkedListTraverse)**

Step 1: Start from head.

Step 2: While current != NULL:

- Print current->name.

- Move to next node.

Step 3: End.

**2. Stack Operations**

**a) Push (stackPush)**

Step 1: Create a new stack node.

Step 2: Store current folder name and copy of folder contents.

Step 3: Set newNode->next = top.

Step 4: Move top = newNode.

**b) Pop (stackPop)**

Step 1: If top == NULL, return failure.

Step 2: Store top->folderName and contents.

Step 3: Move top = top->next.

Step 4: Free old top node.

Step 5: Return success.

**c) IsEmpty (stackIsEmpty)**

Step 1: If top == NULL return true, else false.

# 9.sample input and output

Current Folder: root

Menu:

1. Navigate to a new folder

2. Go Back

3. Go Forward

4. Add a file/folder

5. Delete a file/folder

6. Show folder contents

7. Exit

Enter your choice: 6

Contents of 'root':

- Documents

- Pictures

- file1.txt

Enter your choice: 1

Enter folder name to navigate: Documents

Navigated to folder 'Documents'.

Current Folder: Documents

Menu:

1. Navigate to a new folder

2. Go Back

3. Go Forward

4. Add a file/folder

5. Delete a file/folder

6. Show folder contents

7. Exit

Enter your choice: 6

Contents of 'Documents':

(empty)

Enter your choice: 4

Enter file/folder name to add: project.docx

'project.docx' added to 'Documents'.

Enter your choice: 6

Contents of 'Documents':

- project.docx

Enter your choice: 2

Moved back to 'root'.

Current Folder: root

Menu:

1. Navigate to a new folder

2. Go Back

3. Go Forward

4. Add a file/folder

5. Delete a file/folder

6. Show folder contents

7. Exit

Enter your choice: 3

Moved forward to 'Documents'.

Current Folder: Documents

Menu:

Enter your choice: 6

Contents of 'Documents':

- project.docx

Enter your choice: 5

Enter file/folder name to delete: project.docx

'project.docx' deleted from 'Documents'.

Enter your choice: 6

Contents of 'Documents':

(empty)

Enter your choice: 7

Exiting program.

# 10. Future Scope and Enhancements

🡪Incorporate support for file creation, deletion, and management alongside directories

🡪Develop a graphical user interface (GUI) to enhance user experience

🡪Implement persistent storage mechanisms to save the filesystem state across sessions

🡪Introduce user authentication and access control to simulate multi-user environments

🡪Implement command history, autocomplete features, and scripting capabilities

# 11. Conclusion

This project successfully models the foundational principles of file system navigation through the application of tree data structures and recursion, while facilitating command parsing and directory operations. It serves as a valuable educational tool, offering practical insights into filesystem design and command-line interface interactions.

# 12. References

- Galvin, P. B., & Gagne, G. (Operating System Concepts).

-Official documentation of Python / C++ / Java programming languages.

-Geeks for Geeks articles on Tree Data Structures and File System Simulation.

-Community discussions on Stack Overflow and GitHub repositories related to filesystem projects.