CAPSTONE PROJECT REPORT

Title: *Smart Linux File Explorer with Activity Logger*

Name: SWAROOP KUMAR SATAPATHY
Registration No.: 2241019345

1. Introduction

This project, titled **Smart Linux File Explorer**, is a command-line based file management system developed using **Modern C++ (C++17)**.
The tool enables users to perform essential file operations such as listing, navigating, creating, deleting, copying, and searching files within a Linux terminal environment.

This enhanced version includes a **novelty feature – an Activity Logger**, which records every operation performed into a file called `activity_log.txt`.
Each action is timestamped, providing transparency and traceability for all file management activities.

2. Objectives

- Design and implement a command-line file management system in C++.
- Apply the concepts of file handling and directory traversal using the <filesystem> library.
- Gain practical knowledge of system-level programming.
- Improve understanding of command-line interfaces and user interaction.

3. Project Description

The project accepts simple text commands such as ls, cd, create, delete, copy, and search to perform file operations.
It provides meaningful feedback to users for success or failure of commands.

4. Commands Implemented

Command	Function	Description
`ls`	List Files	Displays all files and directories in the current working directory

`cd <folder>`	Change Directory	Moves the user to the specified folder
`create <filename>`	Create File	Creates a new file in the current directory
`delete <filename>`	Delete File	Removes a file from the system
`copy <src> <dest>`	Copy File	Copies one file to another location
`search <keyword>`	Search	Finds files matching the given keyword
`exit`	Exit Program	Closes the application gracefully

4. System Requirements

Software Requirements:

Linux (Ubuntu / Fedora / WSL)

GCC compiler with C++17 support

Code::Blocks / Visual Studio Code

Hardware Requirements:

Intel i3 or higher processor

Minimum 2 GB RAM

100 MB free disk space

5. Implementation Details

Programming Language:

C++ (C++17 Standard)

Key Libraries Used

- `<filesystem>` — For file and directory handling

- `<fstream>` — For file I/O operations
- `<sstream>` — For parsing user input
- `<ctime>` — For generating timestamps in logs

6. Output and Testing

The program was tested on Linux systems and produced correct outputs for various file operations.
Error handling ensures the program remains stable even for invalid inputs.
Screenshots of successful execution are attached at the end of this report.

7. Learning Outcomes

Gained practical experience with Linux file systems.

Learned to use the C++17 <filesystem> library.

Improved knowledge of file handling and error management.

Understood terminal-based programming and input handling.

8.Code

```
#include <iostream>
#include <filesystem>
#include <fstream>
#include <sstream>
#include <ctime>

using namespace std;
namespace fs = std::filesystem;

void logAction(string action) {
    ofstream log("activity_log.txt", ios::app);
    if (log.is_open()) {
        time_t now = time(0);
        char *dt = ctime(&now);
        string timeStr(dt);
        timeStr.pop_back();
        log << "[" << timeStr << "] " << action << endl;
    }
    log.close();
```

```
}

void listFiles() {
    cout << "\nFiles and Directories:\n";
    for (auto &entry : fs::directory_iterator(fs::current_path()))
{
        if (entry.is_directory())
            cout << "[DIR] " << entry.path().filename().string()
<< endl;
        else
            cout << "      " << entry.path().filename().string()
<< endl;
    }
    logAction("Listed files in " + fs::current_path().string());
}

void changeDir(string folder) {
    try {
        fs::current_path(folder);
        cout << "Now in: " << fs::current_path() << endl;
        logAction("Changed directory to: " +
fs::current_path().string());
    } catch (...) {
        cout << "Directory not found.\n";
        logAction("Failed to change directory: " + folder);
    }
}

void createFile(string filename) {
    ofstream file(filename);
    if (file.is_open()) {
        cout << "File created: " << filename << endl;
        logAction("Created file: " + filename);
    } else {
        cout << "Error creating file.\n";
        logAction("Failed to create file: " + filename);
    }
    file.close();
}

void deleteFile(string filename) {
    if (fs::remove(filename)) {
```

```
        cout << "File deleted: " << filename << endl;
        logAction("Deleted file: " + filename);
    } else {
        cout << "File not found.\n";
        logAction("Failed to delete file: " + filename);
    }
}

void copyFile(string src, string dest) {
    try {
        fs::copy_file(src, dest,
fs::copy_options::overwrite_existing);
        cout << "Copied " << src << " to " << dest << endl;
        logAction("Copied file from " + src + " to " + dest);
    } catch (...) {
        cout << "Error copying file.\n";
        logAction("Failed to copy " + src + " to " + dest);
    }
}

void searchFile(string key) {
    cout << "Search Results:\n";
    for (auto &entry :
fs::recursive_directory_iterator(fs::current_path())) {
        if (entry.path().filename().string().find(key) !=
string::npos)
            cout << entry.path().string() << endl;
    }
    logAction("Searched for: " + key);
}

int main() {
    cout << "===== SMART LINUX FILE EXPLORER =====\n";
    cout << "Current Directory: " << fs::current_path() << endl;
    logAction("Program started.");

    string cmd, arg1, arg2;

    while (true) {
        cout << "\n>>> ";
        getline(cin, cmd);
```

```
        stringstream ss(cmd);
        ss >> cmd >> arg1 >> arg2;

        if (cmd == "ls")
            listFiles();
        else if (cmd == "cd")
            changeDir(arg1.empty() ? "." : arg1);
        else if (cmd == "create")
            createFile(arg1);
        else if (cmd == "delete")
            deleteFile(arg1);
        else if (cmd == "copy")
            copyFile(arg1, arg2);
        else if (cmd == "search")
            searchFile(arg1);
        else if (cmd == "exit") {
            cout << "Goodbye!\n";
            logAction("Program exited.");
            break;
        } else {
            cout << "Invalid command.\n";
        }
    }

    return 0;
}
```

9. Novelty

The project introduces a **new feature – Activity Logger** that:

- Records every file operation in real-time.
- Saves actions in a separate log file `activity_log.txt`.
- Each log entry includes a **timestamp** and **description of the operation**.

This enhancement provides auditing, transparency, and a professional edge to the traditional file explorer concept.

10.Error Handling

All operations use try-catch blocks to handle invalid directories, missing files, and permission issues.
Invalid commands are gracefully handled by displaying “Invalid command.”

11. Conclusion

The Linux File Explorer successfully demonstrates integration of Modern C++ with Linux system programming.
It allows performing core file operations efficiently from the terminal.
The project meets its educational objective by providing hands-on experience with C++ filesystem operations and Linux OS concepts.

12. Future Enhancements

- Folder creation and deletion options (`mkdir`, `rmdir`).
- Menu-driven interface for easier navigation.
- Integration with file compression utilities (.zip/.tar).
- GUI version using the Qt framework.
- Real-time display of log file updates.

13. References

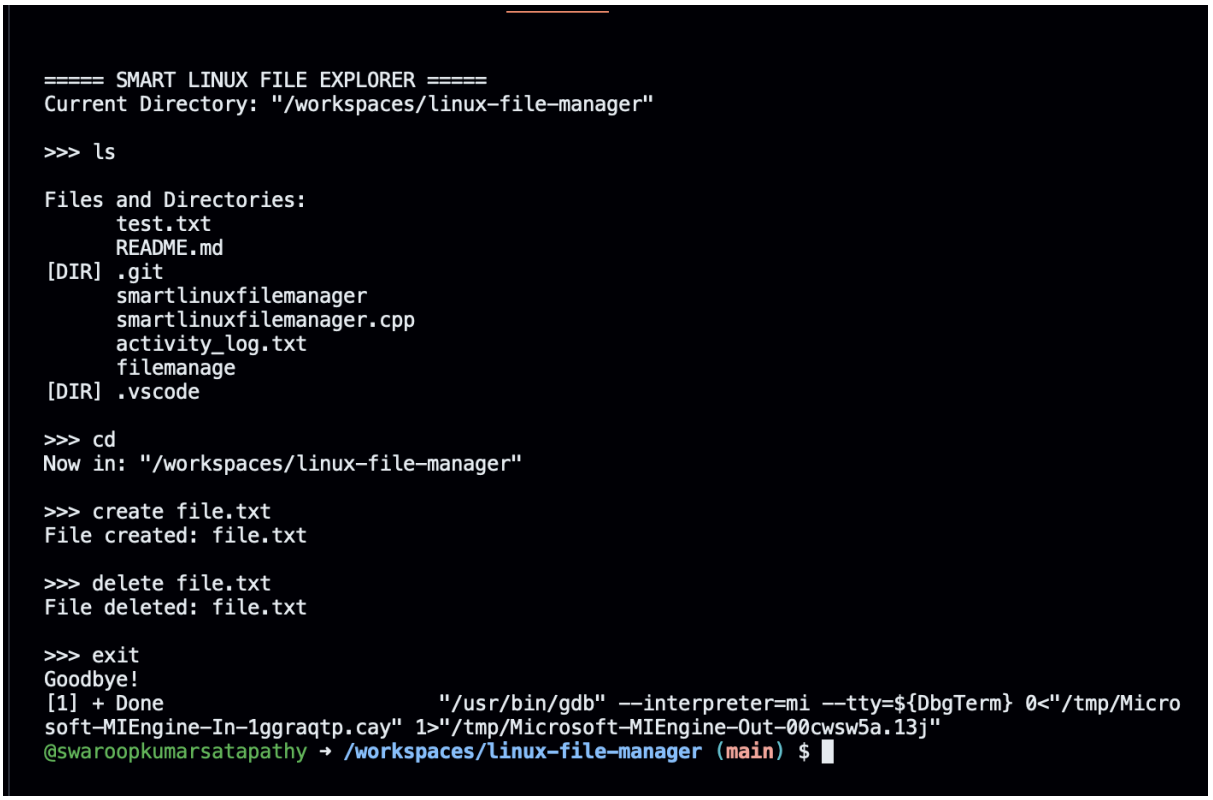
C++ Filesystem Reference

GNU GCC Documentation

Linux Manual Pages (man filesystem, man g++)

W3Schools C++ File Handling Tutorials

Appendix: Screenshots



Name: SWAROOP KUMAR SATAPATHY
Registration No.: 2241019345
Section: 2241024
Wipro Batch: 3

End of Report