File Explorer in C++ -> Project Report

Arkesh Pritam

1.Introduction

File management is one of the core functionalities of any operating system. This project, **File Explorer using C++**, provides a command-line interface that allows users to perform file operations such as creating, deleting, copying, moving, searching, and viewing permissions. The application directly uses **UNIX system calls** such as `open()`, `stat()`, `chmod()`, `opendir()`, and `readdir()` for low-level file handling.

The project demonstrates how file systems work internally and helps understand OS concepts like directories, file permissions, and file I/O operations.

2.Objective

The main objectives of this project are:

- To implement a simple file explorer in C++.
- To understand and use UNIX/Linux system calls for file handling.
- To perform essential file operations:
    - List directory
    - Change directory
    - Create/Delete file
    - Move/Rename file

    - Copy file
    - Search file
    - Show/modify file permissions
-
- To build an interactive menu-based console application.
- To apply C/C++ standard libraries and POSIX APIs.

3.Technologies Used

Component	Description
Language	C++ (Modern C++ with basic STL)
System Calls	`opendir`, `readdir`, `chdir`, `chmod`, `stat`, `remove`, `rename`
Libraries	`<iostream>`, `<dirent.h>`, `<unistd.h>`, `<sys/stat.h>`, `<cstdio>`, `<cstring>`
Operating System	Linux/Unix environment

4.Implementation Details

- **File operations** using C standard library (`fopen`, `fread`, `fwrite`)
- **Permission management** using POSIX stat structures
- **Menu-driven interaction** using loops and user input

The program begins by fetching the current working directory and then repeatedly shows the menu until the user selects Exit.

5.Code

```
#include <iostream>
#include <dirent.h>
#include <unistd.h>
#include <sys/stat.h>
#include <cstring>
#include <vector>
#include <cstdio>
using namespace std;

// List directory contents
void listDirectory(const char* path) {
        DIR *dir = opendir(path);
        if (!dir) {
        cout << "Cannot open directory: " << path << endl;
        return;
        }
        struct dirent *entry;
        cout << "Contents of: " << path << endl;
        while ((entry = readdir(dir)) != NULL) {
        cout << entry->d_name << endl;
        }
        closedir(dir);
}

// Change directory
bool changeDirectory(const char* path) {
        return chdir(path) == 0;
}

// Create file
void createFile(const char* filename) {
        FILE *f = fopen(filename, "w");
        if (f) {
        fprintf(f, "New file created");
        fclose(f);
        cout << "File created: " << filename << endl;
        } else {
        cout << "Cannot create file: " << filename << endl;
        }
}

// Delete file
void deleteFile(const char* filename) {
        if (remove(filename) == 0) {
```

```
        cout << "Deleted file: " << filename << endl;
        } else {
        cout << "Cannot delete file: " << filename << endl;
        }
}

// Rename file
void moveFile(const char* source, const char* dest) {
        if (rename(source, dest) == 0) {
        cout << "Renamed " << source << " to " << dest << endl;
        } else {
        cout << "Cannot rename " << source << endl;
        }
}

// Copy file
void copyFile(const char* src, const char* dst) {
        FILE *in = fopen(src, "r");
        if (!in) { cout << "Cannot open source file.\n"; return; }
        FILE *out = fopen(dst, "w");
        if (!out) { cout << "Cannot create destination file.\n"; fclose(in); return; }
        char buffer[4096];
        size_t bytes;
        while ((bytes = fread(buffer, 1, sizeof(buffer), in)) > 0)
        fwrite(buffer, 1, bytes, out);
        fclose(in);
        fclose(out);
        cout << "Copied " << src << " to " << dst << endl;
}

// Search for file in directory (non-recursive)
void searchFile(const char* path, const char* filename) {
        DIR *dir = opendir(path);
        if (!dir) { cout << "Cannot open directory for search.\n"; return; }
        struct dirent *entry;
        bool found = false;
        while ((entry = readdir(dir)) != NULL) {
        if (strstr(entry->d_name, filename)) {
        cout << "Found: " << entry->d_name << endl;
        found = true;
        }
        }
        if (!found) cout << "File not found in " << path << endl;
        closedir(dir);
}

// Show permissions
void showPermissions(const char* filename) {
```

```
        struct stat info;
        if (stat(filename, &info) == 0) {
        cout << "Permissions for " << filename << ": ";
        cout << ((info.st_mode & S_IRUSR) ? "r" : "-")
        << ((info.st_mode & S_IWUSR) ? "w" : "-")
        << ((info.st_mode & S_IXUSR) ? "x" : "-") << endl;
        } else
        cout << "File does not exist\n";
}

// Change to read-only
void setReadOnly(const char* filename) {
        if (chmod(filename, S_IRUSR | S_IRGRP | S_IROTH) == 0) {
        cout << "Changed " << filename << " to read-only\n";
        } else {
        cout << "Failed to change permissions\n";
        }
}

int main() {
        char cwd[1024];
        getcwd(cwd, sizeof(cwd));
        cout << "Starting in directory: " << cwd << endl;
        int choice;
        do {
        cout << "\nFile Explorer Menu:\n";
        cout << "1. List Directory\n";
        cout << "2. Change Directory\n";
        cout << "3. Create File\n";
        cout << "4. Delete File\n";
        cout << "5. Rename File\n";
        cout << "6. Copy File\n";
        cout << "7. Search File\n";
        cout << "8. Show Permissions\n";
        cout << "9. Set Read-Only Permissions\n";
        cout << "0. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;
        cin.ignore();
        if (choice == 1) {
        getcwd(cwd, sizeof(cwd));
        listDirectory(cwd);
        } else if (choice == 2) {
        char path[1024];
        cout << "Enter directory path: ";
        cin.getline(path, sizeof(path));
        if (changeDirectory(path))
                cout << "Changed directory successfully.\n";
```

```
        else
                cout << "Failed to change directory.\n";
        } else if (choice == 3) {
        char filename[1024];
        cout << "Enter new file name: ";
        cin.getline(filename, sizeof(filename));
        createFile(filename);
        } else if (choice == 4) {
        char filename[1024];
        cout << "Enter file name to delete: ";
        cin.getline(filename, sizeof(filename));
        deleteFile(filename);
        } else if (choice == 5) {
        char source[1024], dest[1024];
        cout << "Enter source file name: ";
        cin.getline(source, sizeof(source));
        cout << "Enter destination file name: ";
        cin.getline(dest, sizeof(dest));
        moveFile(source, dest);
        } else if (choice == 6) {
        char source[1024], dest[1024];
        cout << "Enter source file name: ";
        cin.getline(source, sizeof(source));
        cout << "Enter destination file name: ";
        cin.getline(dest, sizeof(dest));
        copyFile(source, dest);
        } else if (choice == 7) {
        char filename[1024];
        cout << "Enter file name to search: ";
        cin.getline(filename, sizeof(filename));
        getcwd(cwd, sizeof(cwd));
        searchFile(cwd, filename);
        } else if (choice == 8) {
        char filename[1024];
        cout << "Enter file name to show permissions: ";
        cin.getline(filename, sizeof(filename));
        showPermissions(filename);
        } else if (choice == 9) {
        char filename[1024];
        cout << "Enter file name to set read-only: ";
        cin.getline(filename, sizeof(filename));
        setReadOnly(filename);
        }
        } while (choice != 0);
        cout << "Exiting File Explorer.\n";
        return 0;
}
```

6.Module Description

List Directory

- Uses `opendir()` and `readdir()` to read all files/folders.
- Displays the contents of the current working directory.

Change Directory

- Uses `chdir()` to change the current directory.
- Updates working path with `getcwd()`.

Create File

- Uses `fopen(filename, "w")` to create a new file.
- Writes default content: `"New file created"`.

Delete File

- Uses `remove()` to delete a file.
- Shows success or failure message.

Move / Rename File

- Uses `rename(source, destination)` to move or rename a file.

Copy File

- Opens source file in read mode and destination in write mode.
- Copies data using buffer of 4096 bytes.

Search File (Non-Recursive)

- Reads contents of directory using `readdir()`.
- Finds matches using `strstr()`.

Show Permissions

- Uses `stat()` to fetch file permission bits.
- Extracts:
    - Read / Write / Execute for owner

Set Read-Only Permissions

- Uses `chmod()` with:
  `S_IRUSR | S_IRGRP | S_IROTH`
  (owner, group, others: read only)

7.Outputs

(a) - list directories

```
arkesh@arkesh-ubuntu:~/File_Explorer$ g++ file_explorer.cpp -o file_explorer
arkesh@arkesh-ubuntu:~/File_Explorer$ ./file_explorer
Starting in directory: /home/arkesh/File_Explorer

File Explorer Menu:
1. List Directory
2. Change Directory
3. Create File
4. Delete File
5. Rename File
6. Copy File
7. Search File
8. Show Permissions
9. Set Read-Only Permissions
0. Exit
Enter choice: 1
Contents of: /home/arkesh/File_Explorer
file_explorer
..
file3
.
file_explorer.cpp
```

(b) – create file

```
arkesh@arkesh-ubuntu:~/File_Explorer$ g++ file_explorer.cpp -o file_explorer
arkesh@arkesh-ubuntu:~/File_Explorer$ ./file_explorer
Starting in directory: /home/arkesh/File_Explorer

File Explorer Menu:
1. List Directory
2. Change Directory
3. Create File
4. Delete File
5. Rename File
6. Copy File
7. Search File
8. Show Permissions
9. Set Read-Only Permissions
0. Exit
Enter choice: 3
Enter new file name: newfile
File created: newfile
```

(c) – delete file

```
arkesh@arkesh-ubuntu:~/File_Explorer$ g++ file_explorer.cpp -o file_explorer
arkesh@arkesh-ubuntu:~/File_Explorer$ ./file_explorer
Starting in directory: /home/arkesh/File_Explorer

File Explorer Menu:
1. List Directory
2. Change Directory
3. Create File
4. Delete File
5. Rename File
6. Copy File
7. Search File
8. Show Permissions
9. Set Read-Only Permissions
0. Exit
Enter choice: 4
Enter file name to delete: newfile
Deleted file: newfile
```

(d) – rename file

```
arkesh@arkesh-ubuntu:~/File_Explorer$ g++ file_explorer.cpp -o file_explorer
arkesh@arkesh-ubuntu:~/File_Explorer$ ./file_explorer
Starting in directory: /home/arkesh/File_Explorer

File Explorer Menu:
1. List Directory
2. Change Directory
3. Create File
4. Delete File
5. Rename File
6. Copy File
7. Search File
8. Show Permissions
9. Set Read-Only Permissions
0. Exit
Enter choice: 5
Enter source file name: file3
Enter destination file name: renamedfile
Renamed file3 to renamedfile
```

(e) – search file

```
arkesh@arkesh-ubuntu:~/File_Explorer$ g++ file_explorer.cpp -o file_explorer
arkesh@arkesh-ubuntu:~/File_Explorer$ ./file_explorer
Starting in directory: /home/arkesh/File_Explorer

File Explorer Menu:
1. List Directory
2. Change Directory
3. Create File
4. Delete File
5. Rename File
6. Copy File
7. Search File
8. Show Permissions
9. Set Read-Only Permissions
0. Exit
Enter choice: 7
Enter file name to search: renamedfile
Found: renamedfile
```

(f) – show permissions

```
arkesh@arkesh-ubuntu:~/File_Explorer$ g++ file_explorer.cpp -o file_explorer
arkesh@arkesh-ubuntu:~/File_Explorer$ ./file_explorer
Starting in directory: /home/arkesh/File_Explorer

File Explorer Menu:
1. List Directory
2. Change Directory
3. Create File
4. Delete File
5. Rename File
6. Copy File
7. Search File
8. Show Permissions
9. Set Read-Only Permissions
0. Exit
Enter choice: 8
Enter file name to show permissions: renamedfile
Permissions for renamedfile: rw-
```

(g) – set read only permissions

```
arkesh@arkesh-ubuntu:~/File_Explorer$ g++ file_explorer.cpp -o file_explorer
arkesh@arkesh-ubuntu:~/File_Explorer$ ./file_explorer
Starting in directory: /home/arkesh/File_Explorer

File Explorer Menu:
1. List Directory
2. Change Directory
3. Create File
4. Delete File
5. Rename File
6. Copy File
7. Search File
8. Show Permissions
9. Set Read-Only Permissions
0. Exit
Enter choice: 9
Enter file name to set read-only: renamedfile
Changed renamedfile to read-only
```

8.Advantages

- Demonstrates use of low-level UNIX system calls.
- Works efficiently with large directories.
- Helps understand file structure and permissions in Linux.
- Simple and easy to implement

9.Limitations

- No support for directories in file copying.
- No GUI interface (command-line only).
- No exception handling for permission-denied errors.
- Search is non recursive (only searches current directory).

10.Conclusion

This project successfully demonstrates a functional file explorer built with C++ and UNIX system calls. It gives the user hands-on understanding of file handling at the system level and covers major operations required in file management.

The application is modular, extensible, and forms a strong foundation for more advanced file system tools or GUI-based explorers.