File Explorer Application using C++ in Linux OS

1.OBJECTIVE:

The main objective of this project is to design and implement a console-based File Explorer in C++ that performs file management operations in a Linux environment. This project demonstrates how system-level programming can be used to interact with the Linux file system through various system calls and C++ functionalities.

Overview:

The File Explorer Application allows the user to perform essential file operations such as:
- Listing files and directories
- Copying, moving, deleting, and creating files
- Searching for files recursively
- Viewing and changing file permissions

The application uses Linux system calls like opendir(), readdir(), stat(), and chmod() to directly interact with the file system. It acts as a simplified command-line tool that mimics some fundamental Linux shell operations within a C++ program.

Key Features:

- List Files: Display all files and directories in the current folder.
- Copy File: Copy data from a source file to a destination file.
- Move File: Relocate or rename files.
- Delete File: Permanently remove files from the directory.
- Create File: Generate a new file in the current directory.
- Search File: Recursively search for a file or folder name.
- View Permissions: Display file permissions in symbolic format (e.g., rwxr-xr-x).
- Change Permissions: Modify file permissions using numeric octal values (e.g., 755, 644).

Tools and Technologies Used:

Language: C++
Compiler: g++ (GNU Compiler Collection)
Operating System: Linux / Ubuntu / WSL
Libraries Used:
- <iostream>
- <fstream>
- <dirent.h>
- <sys/stat.h>
- <unistd.h>
- <iomanip>
- <string>

2.CODE:

```
#include <cstring>
#include <string>
#include <iomanip>
#include <dirent.h>
#include <sys/stat.h>
#include <unistd.h>
using namespace std;

void listFiles(const char* path) {
   DIR* dir = opendir(path);
   if (!dir) {
      perror("opendir");
      return;
   }
   struct dirent* entry;
   while ((entry = readdir(dir)) != NULL) {
      cout << entry->d_name << endl;
   }
   closedir(dir);
}
void copyFile(string source, string destination) {
   ifstream src(source, ios::binary);
   ofstream dest(destination, ios::binary);
   if (!src) {
      cout << "Source file not found!\n";
      return;
   }
   dest << src.rdbuf();
   cout << "File copied successfully!\n";
}

void moveFile(string oldPath, string newPath) {
   if (rename(oldPath.c_str(), newPath.c_str()) == 0)
      cout << "File moved successfully!\n";
   else
      perror("Error moving file");
}

void deleteFile(string filename) {
   if (remove(filename.c_str()) == 0)
      cout << "File deleted successfully!\n";
   else
      perror("Error deleting file");
}

void createFile(string filename) {
   ofstream newFile(filename);
   if (newFile) {
```

```
        cout << "File created successfully!\n";
        newFile.close();
    } else {
        cout << "Error creating file!\n";
    }
}
void searchFile(const string &basePath, const string &target) {
    DIR *dir = opendir(basePath.c_str());
    if (!dir) return;

    struct dirent *entry;
    struct stat info;

    while ((entry = readdir(dir)) != NULL) {
        string name = entry->d_name;

        // Skip current and parent folders
        if (name == "." || name == "..")
            continue;

        string fullPath = basePath + "/" + name;

        // Check if it's a directory
        if (stat(fullPath.c_str(), &info) == -1) {
                continue;
        }

        if (S_ISDIR(info.st_mode)) {
            // Recursive call to search inside subdirectories
            searchFile(fullPath, target);
        }

        // Check if file/folder name contains the target
        if (name.find(target) != string::npos) {
            cout << "Found: " << fullPath << endl;
        }
    }

    closedir(dir);
}
// Convert mode (st_mode) to rwxr-xr-x style string
string permissionString(mode_t mode) {
    string s;
    if (S_ISDIR(mode)) s += 'd'; else s += '-';

    s += (mode & S_IRUSR) ? 'r' : '-';
    s += (mode & S_IWUSR) ? 'w' : '-';
    s += (mode & S_IXUSR) ? 'x' : '-';
    s += (mode & S_IRGRP) ? 'r' : '-';
    s += (mode & S_IWGRP) ? 'w' : '-';
```

```
    s += (mode & S_IXGRP) ? 'x' : '-';
    s += (mode & S_IROTH) ? 'r' : '-';
    s += (mode & S_IWOTH) ? 'w' : '-';
    s += (mode & S_IXOTH) ? 'x' : '-';

    return s;
}

void viewPermissions(const string &path) {
    struct stat info;
    if (stat(path.c_str(), &info) == -1) {
        perror(("stat: " + path).c_str());
        return;
    }
    cout << permissionString(info.st_mode) << "  ";
    cout << setw(8) << info.st_size << " bytes  ";
    cout << path << endl;
}

void changePermissions(const string &path, const string &modeStr) {
    int modeInt = 0;
    try {
        modeInt = stoi(modeStr, nullptr, 8); // base 8
    } catch (...) {
        cout << "Invalid mode format. Use octal like 644 or 0755.\n";
        return;
    }
    mode_t mode = static_cast<mode_t>(modeInt);

    if (chmod(path.c_str(), mode) == 0) {
        cout << "Permissions changed successfully.\n";
        viewPermissions(path);
    } else {
        perror(("chmod: " + path).c_str());
        cout << "You may need appropriate privileges to change these permissions.\n";
    }
}

int main() {
    string command;
    while (true) {
        cout << "\nAvailable commands: list | copy | move | delete | create | search | perm |
chmod | exit\n";
        cout << "Enter command: ";
        cin >> command;

        if (command == "list") {
            listFiles(".");
        }
        else if (command == "copy") {
```

```
    string src, dest;
    cout << "Enter source file: ";
    cin >> src;
    cout << "Enter destination file: ";
    cin >> dest;
    copyFile(src, dest);
}
 else if (command == "move") {
    string src, dest;
    cout << "Enter source file: ";
    cin >> src;
    cout << "Enter destination file: ";
    cin >> dest;
    moveFile(src, dest);
}
 else if (command == "delete") {
    string filename;
    cout << "Enter file to delete: ";
    cin >> filename;
    deleteFile(filename);
}
 else if (command == "create") {
    string filename;
    cout << "Enter new file name: ";
    cin >> filename;
    createFile(filename);
}
 else if (command == "search") {
    string keyword;
          cout << "Enter file name or keyword to search: ";
             cin >> keyword;
          searchFile(".", keyword);
          }
          else if (command == "perm") {
          string filename;
          cout << "Enter file or path to view permissions: ";
          cin >> filename;
          viewPermissions(filename);
          }
          else if (command == "chmod") {
          string filename, mode;
          cout << "Enter file or path to change permissions: ";
          cin >> filename;
          cout << "Enter numeric mode (e.g. 644 or 0755): ";
          cin >> mode;
          changePermissions(filename, mode);
         }
else if (command == "exit") {
    cout << "Exiting File Explorer...\n";
    break;
```

```
    }
    else {
        cout << "Invalid command!\n";
    }
  }
  return 0;
}
```

3.FULL SCREENSHOTS:

Code:

```cpp
#include <iostream>
#include <unistd.h>
#include <fstream>
#include <cstring>
#include <string>
#include <iomanip>
#include <dirent.h>
#include <sys/stat.h>
#include <unistd.h>
using namespace std;

void listFiles(const char* path) {
    DIR* dir = opendir(path);
    if (!dir) {
        perror("opendir");
        return;
    }
    struct dirent* entry;
    while ((entry = readdir(dir)) != NULL) {
        cout << entry->d_name << endl;
    }
    closedir(dir);
}
void copyFile(string source, string destination) {
    ifstream src(source, ios::binary);
    ofstream dest(destination, ios::binary);
    if (!src) {
        cout << "Source file not found!\n";
        return;
    }
    dest << src.rdbuf();
    cout << "File copied successfully!\n";
}
```

```cpp
void moveFile(string oldPath, string newPath) {
    if (rename(oldPath.c_str(), newPath.c_str()) == 0)
        cout << "File moved successfully!\n";
    else
        perror("Error moving file");
}

void deleteFile(string filename) {
    if (remove(filename.c_str()) == 0)
        cout << "File deleted successfully!\n";
    else
        perror("Error deleting file");
}

void createFile(string filename) {
    ofstream newFile(filename);
    if (newFile) {
        cout << "File created successfully!\n";
        newFile.close();
    } else {
        cout << "Error creating file!\n";
    }
}
void searchFile(const string &basePath, const string &target) {
    DIR *dir = opendir(basePath.c_str());
    if (!dir) return;

    struct dirent *entry;
    struct stat info;
```

```cpp
    while ((entry = readdir(dir)) != NULL) {
        string name = entry->d_name;

        // Skip current and parent folders
        if (name == "." || name == "..")
            continue;

        string fullPath = basePath + "/" + name;

        // Check if it's a directory
        if (stat(fullPath.c_str(), &info) == -1) {
        continue;
        }

        if (S_ISDIR(info.st_mode)) {
            // Recursive call to search inside subdirectories
            searchFile(fullPath, target);
        }

        // Check if file/folder name contains the target
        if (name.find(target) != string::npos) {
            cout << "Found: " << fullPath << endl;
        }
    }

    closedir(dir);
}
// Convert mode (st_mode) to rwxr-xr-x style string
string permissionString(mode_t mode) {
    string s;
    if (S_ISDIR(mode)) s += 'd'; else s += '-';
```

```
    s += (mode & S_IRUSR) ? 'r' : '-';
    s += (mode & S_IWUSR) ? 'w' : '-';
    s += (mode & S_IXUSR) ? 'x' : '-';
    s += (mode & S_IRGRP) ? 'r' : '-';
    s += (mode & S_IWGRP) ? 'w' : '-';
    s += (mode & S_IXGRP) ? 'x' : '-';
    s += (mode & S_IROTH) ? 'r' : '-';
    s += (mode & S_IWOTH) ? 'w' : '-';
    s += (mode & S_IXOTH) ? 'x' : '-';

    return s;
}

void viewPermissions(const string &path) {
    struct stat info;
    if (stat(path.c_str(), &info) == -1) {
        perror(("stat: " + path).c_str());
        return;
    }
    cout << permissionString(info.st_mode) << "  ";
    cout << setw(8) << info.st_size << " bytes  ";
    cout << path << endl;
}

void changePermissions(const string &path, const string &modeStr) {
    int modeInt = 0;
    try {
        modeInt = stoi(modeStr, nullptr, 8); // base 8
    } catch (...) {
        cout << "Invalid mode format. Use octal like 644 or 0755.\n";
        return;
    }
    mode_t mode = static_cast<mode_t>(modeInt);
    if (chmod(path.c_str(), mode) == 0) {
        cout << "Permissions changed successfully.\n";
        viewPermissions(path);
    } else {
        perror(("chmod: " + path).c_str());
        cout << "You may need appropriate privileges to change these permissions.\n";
    }
}

int main() {
    string command;
    while (true) {
        cout << "\nAvailable commands: list | copy | move | delete | create | search | perm | chmod | exit\n";
        cout << "Enter command: ";
        cin >> command;

        if (command == "list") {
            listFiles(".");
        }
        else if (command == "copy") {
            string src, dest;
            cout << "Enter source file: ";
            cin >> src;
            cout << "Enter destination file: ";
            cin >> dest;
            copyFile(src, dest);
        }
        else if (command == "move") {
            string src, dest;
            cout << "Enter source file: ";
            cin >> src;
            cout << "Enter destination file: ";
            cin >> dest;
            moveFile(src, dest);
```

```
        }
        else if (command == "delete") {
            string filename;
            cout << "Enter file to delete: ";
            cin >> filename;
            deleteFile(filename);
        }
        else if (command == "create") {
            string filename;
            cout << "Enter new file name: ";
            cin >> filename;
            createFile(filename);
        }
        else if (command == "search") {
            string keyword;
            cout << "Enter file name or keyword to search: ";
            cin >> keyword;
        searchFile(".", keyword);
        }
        else if (command == "perm") {
            string filename;
            cout << "Enter file or path to view permissions: ";
            cin >> filename;
            viewPermissions(filename);
        }
        else if (command == "chmod") {
            string filename, mode;
            cout << "Enter file or path to change permissions: ";
            cin >> filename;
            cout << "Enter numeric mode (e.g. 644 or 0755): ";
            cin >> mode;
            changePermissions(filename, mode);
          }
          else if (command == "exit") {
                cout << "Exiting File Explorer...\n";
                break;
          }
          else {
                cout << "Invalid command!\n";
          }
    }
    return 0;
}
```

Output:

```
Available commands: list | copy | move | delete | create | search | perm | chmod | exit
Enter command: list
.
..
.cache
.eclipse
.p2
.packettracer
.vscode
3D Objects
AppData
Application Data
```

```
Available commands: list | copy | move | delete | create | search | perm | chmod | exit
Enter command: copy
Enter source file: soa.txt
Enter destination file: imp.cpp
File copied successfully!

Available commands: list | copy | move | delete | create | search | perm | chmod | exit
Enter command: move
Enter source file: imp.cpp
Enter destination file: soa.txt
File moved successfully!

Available commands: list | copy | move | delete | create | search | perm | chmod | exit
Enter command: delete
Enter file to delete: hope
File deleted successfully!

Available commands: list | copy | move | delete | create | search | perm | chmod | exit
Enter command: create
Enter new file name: 2241016484.cpp
File created successfully!

Available commands: list | copy | move | delete | create | search | perm | chmod | exit
Enter command: perm
Enter file or path to view permissions: soa.txt
-rwxrwxrwx         0 bytes  soa.txt

Available commands: list | copy | move | delete | create | search | perm | chmod | exit
Enter command: chmod
Enter file or path to change permissions: soa.txt
Enter numeric mode (e.g. 644 or 0755): 644
Permissions changed successfully.
-rwxrwxrwx         0 bytes  soa.txt

Available commands: list | copy | move | delete | create | search | perm | chmod | exit
Enter command: exit
Exiting File Explorer...
```

4.CONCLUSION:

The File Explorer Application successfully demonstrates how C++ can interface with the Linux operating system for performing file and directory management operations. The project uses system calls and standard C++ file I/O techniques to implement a command-driven interface similar to Linux terminal utilities.

Through this project, we gain an understanding of:
- Linux file system handling
- File permission management using symbolic and octal modes
- Recursive directory traversal
- The power of C++ in system-level programming

This project provides a foundational understanding of how file operations are executed at the operating system level.