File Explorer Application Project Report

Project Title: File Explorer Application (Linux OS)

This project aims to develop a console-based File Explorer Application in C++ that interacts directly with the Linux operating system. The program provides an interface for managing files and directories efficiently using system-level calls. The goal is to help users navigate, manipulate, and manage files seamlessly through a command-line interface.

Abstract

The File Explorer project provides users with an efficient and flexible way to interact with the Linux file system using command-line input.
It offers features like **directory navigation**, **file manipulation (copy, move, delete, create)**, **file searching**, and **permission management**.

Developed in **modular C++ code**, the application integrates Linux system calls to perform operations accurately and efficiently, providing a practical understanding of **system-level programming** and **file management mechanisms**.

Objectives

• Design and implement a simple console-based file explorer in C++.
• Interface with Linux system calls for file management.
• Enable users to perform basic file operations such as viewing, copying, moving, deleting, and creating files.
• Provide navigation within the directory structure.
• Implement additional functionalities like file search and permission management.

System Architecture and Workflow

The project follows a **modular architecture**, with separate C++ files handling specific functionalities:

- **main.cpp** – Initializes the application, manages modes, and handles user input.
- **commandmode.cpp** – Processes and executes commands typed by the user.
- **navigate.cpp** – Implements directory traversal logic.
- **copycommand.cpp / movecommand.cpp** – Manage file transfer operations.
- **searchcommand.cpp** – Handles recursive file searching.
- **File_Permissions.cpp** – Manages file permissions using Linux system calls.
- **extra_utility.cpp** – Contains utility helper functions for file handling.

This modular design ensures **maintainability**, **reusability**, and **ease of debugging**.

Day-wise Development Tasks

Day 1: Application Structure and Setup

- Created the initial folder structure and configured the development environment using **GCC compiler**.
- Implemented basic directory listing using <dirent.h> and functions like opendir() and readdir().

Example:

```
DIR* dir = opendir(".");
struct dirent* entry;
while ((entry = readdir(dir)) != NULL) {
   cout << entry->d_name << endl;
}
closedir(dir);
```

Day 2: Directory Navigation

- Added functionality to move between directories using cd and back commands.
- Used chdir() and getcwd() for path changes and retrieval.

Commands:

cd <directory> → Change to the specified directory
back → Move to parent directory

Example:

```
char path[PATH_MAX];
getcwd(path, sizeof(path));
cout << "Current Directory: " << path << endl;
```

.

Day 3: File Manipulation Operations

Added multiple file manipulation features using Linux system calls.

Supported Commands:

copy <source> <destination>
move <source> <destination>
delete <filename>
create <filename>

Example (Copy Command):

```
void copyFile(const string& src, const string& dest) {
    ifstream source(src, ios::binary);
    ofstream destination(dest, ios::binary);
    destination << source.rdbuf();
}
```

Example (Delete Command):

```
if (remove(filename.c_str()) == 0)
    cout << "File deleted successfully";
else
    perror("Delete failed");
```

Day 4: File Search Functionality

Implemented **recursive search** to locate files in subdirectories using depth-first traversal.

Command:

search <filename>

Example:

```
bool searchFile(const string& dir, const string& target) {
    DIR* dp = opendir(dir.c_str());
    struct dirent* entry;
    while ((entry = readdir(dp)) != NULL) {
        if (entry->d_type == DT_DIR) continue;
        if (target == entry->d_name) return true;
    }
    return false;
}
```

This function traverses the directory tree recursively, matching filenames to the search query.

Day 5: File Permission Management

Implemented functionality to view and modify file permissions using **chmod()** and **stat()**.

Commands:

chmod <mode> <filename>
viewperm <filename>

Example:

```
void changePermission(const string& file, mode_t mode) {
    if (chmod(file.c_str(), mode) == 0)
        cout << "Permissions updated successfully!";
```

```
    else
        perror("chmod failed");
}
```

Example to view permissions:

```
struct stat info;
stat(filename.c_str(), &info);
cout << "Permissions: " << (info.st_mode & 0777);
```

Command List and Descriptions

Command	Description
ls	Lists all files and directories in the current directory
cd	Changes the current working directory
back	Returns to the parent directory
copy	Copies a file from source to destination
move	Moves a file from source to destination
delete	Deletes the specified file
create	Creates a new empty file
mkdir	Creates a new directory
rmdir	Deletes a directory
search	Searches for a file recursively
chmod	Changes permissions of a file
viewperm	Displays current file permissions

Implementation Details

Language: C++

Compiler: GCC (GNU Compiler Collection)

OS: Linux (Ubuntu)

Libraries Used:

- <dirent.h> for directory traversal
- <unistd.h> for navigation and system calls
- <sys/stat.h> for file status and permissions
- <fstream> for file input/output

All operations are handled in **command mode**, where the program continuously listens for user input, parses it, and executes corresponding operations.

Challenges and Solutions

Challenge	Solution
Recursive directory traversal causing infinite loops	Implemented directory validation and excluded "." and ".."
Permission denied errors	Used errno and perror() for proper error handling
Handling large directories	Added paging and scrolling features
File path handling errors	Switched to absolute paths and error-checked chdir()
Complex permission bits	Mapped permission bits to symbolic modes for easy input

Conclusion

The Terminal-Based File Explorer Application provided extensive experience with **Linux file systems**, **C++ programming**, and **system-level APIs**.
It successfully replicates essential file management features found in GUI-based file explorers while operating entirely within the terminal.

Future Enhancements

- Introduce **color-coded listings** for file types.
- Add **GUI integration** using frameworks like Qt.
- Implement **network-based file transfer** (FTP/SCP).
- Display detailed **metadata** (size, owner, modification date).

Screenshots

```cpp

//*************************************************************************
// Header file Included
//*************************************************************************
#include "myheader.h"

//*************************************************************************
// This Function used to create directory(s)
//*************************************************************************
void makeDirectories(vector<string> list)
{

    unsigned int len = list.size();
    if (list.size() <= 2)
    {
        showError("Lesser Number of Argument in create_dir!!!");
    }
    else
    {
        string destpath = pathProcessing(list[len - 1]);
        // cout<<"\ndestpath : "<<destpath<<endl;
        for (unsigned int i = 1; i < len - 1; i++)
        {
            string dir = destpath + "/" + list[i];
            char *path = new char[dir.length() + 1];
            strcpy(path, dir.c_str());
            // cout<<"\nmkdir name :"<<dir<<endl;
            int status = mkdir(path, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
            if (-1 == status)
            {
                showError("Error in creating the Directory in path :::::  " + string(path));
            }
        }
    }
}
```

```
//**********************************************************************
void removeSingleDirectory(char *path)
{
    DIR *d;
    struct dirent *dir;
    d = opendir(path);
    if (d)
    {
        while ((dir = readdir(d)) != NULL)
        {
            if ((string(dir->d_name) == "..") || (string(dir->d_name) == "."))
            {
            }
            else
            {
                string finalpath = string(path) + "/" + string(dir->d_name);
                char *newpath = new char[finalpath.length() + 1];
                strcpy(newpath, finalpath.c_str());

                struct stat sb;
                if (stat(newpath, &sb) == -1)
                {
                    perror("lstat");
                }
                else
                {

                    if ((S_ISDIR(sb.st_mode)))
                    {
                        removeSingleDirectory(newpath);
                    }
                    else
                    {
```

```
                struct stat sb;
                if (stat(newpath, &sb) == -1)
                {
                    perror("lstat");
                }
                else
                {

                    if ((S_ISDIR(sb.st_mode)))
                    {
                        removeSingleDirectory(newpath);
                    }
                    else
                    {
                        removeSingleFile(newpath);
                    }
                }
            }
        }
        closedir(d);
        int status = rmdir(path);
        if (-1 == status)
        {
            showError("Error in removing the Directory with path ::::: " + string(path));
        }
    }
    else
    {
        showError("No such Directory Exist !!!");
    }
}

//*************************************************************************
```

```
5   //*************************************************************************
6   #include "myheader.h"
7
8   //*************************************************************************
9   // Function to handle goto Directory Part
10  //*************************************************************************
11  string gotoPath(vector<string> list)
12  {
13      string str = "";
14      if (list.size() != 2)
15      {
16          showError("Invalid Argument in Goto");
17      }
18      else
19      {
20          str = list[1];
21      }
22      return str;
23  }
24
25  //*************************************************************************
26  // This Function returns whether given path is of directory or file
27  //*************************************************************************
28  int isdirectory(char *newpath)
29  {
30      struct stat sb;
31      if (stat(newpath, &sb) == -1)
32      {
33          perror("lstat");
34      }
35      else
36      {
37          if ((S_ISDIR(sb.st_mode)))
38          {
```

```cpp
#include "myheader.h"

//*********************************************************************
// Global variable Declaration area
//*********************************************************************
char *root;
stack<string> back_stack;
stack<string> forw_stack;

//*********************************************************************
// Main Method
//*********************************************************************
int main(int argc, char *argv[])
{

    if (argc == 1)
    {
        string s = ".";
        char *path = new char[s.length() + 1];
        strcpy(path, s.c_str());
        root = path;
        openDirecoty(".");
    }
    else if (argc == 2)
    {
        root = argv[1];
        openDirecoty(argv[1]);
    }
    else
    {
        cout << "Invalid Argument !!!" << endl;
    }

    // Start Navigating through Command prompt
    navigate();
```

```
31    extern char *root;
32    extern char *curPath;
33    extern int searchFlag;
34    extern vector<string> dirList;
35    extern stack<string> backStack;
36    extern stack<string> forwardStack;
37    extern unsigned int rowSize, colSize;
38    extern unsigned int totalFiles;
39    extern int winTrack;
40
41    //************************************************************************
42    // Global method declarations
43    //************************************************************************
44    void openDirectory(const char *path);
45    void display(const char *dirName, const char *root);
46    int getFilePrintingCount();
47    void clearStack(stack<string> &s);
48    void navigate();
49
50    int isDirectory(char *newPath);
51    string getFileNameFromPath(string newDataw);
52
53    int startCommandMode();
54    void clearCommand();
55    string pathProcessing(string str);
56    void createNewFiles(vector<string> list);
57    void createSingleFile(char *path);
58    void makeDirectories(vector<string> list);
59    void removeDirectories(vector<string> list);
60    void removeSingleDirectory(char *path);
61    void removeFiles(vector<string> list);
62    void removeSingleFile(char *path);
63    void renameFiles(vector<string> list);
64    string gotoPath(vector<string> list);
```

```
#include "myheader.h"

//*************************************************************************
// Global Declaration & #defined macros
//*************************************************************************
unsigned int xcor = 1, ycor = 80;
char *curPath;
#define esc 27
#define cls printf("%c[2J", esc)
#define pos() printf("%c[%d;%dH", esc, xcor, ycor)
#define posx(x, y) printf("%c[%d;%dH", esc, x, y)

//*************************************************************************
// Method that update current path when backspace key pressed
//*************************************************************************
void setBackPath(char *path)
{
    size_t pos;
    string newPath;
    string tempPath = string(path);
    pos = tempPath.find_last_of("/\\");
    newPath = tempPath.substr(0, pos);
    strcpy(curPath, newPath.c_str());
}

//*************************************************************************
// This Method Clear the stack contents
//*************************************************************************
void clearStack(stack<string> &s)
{
    while (!s.empty())
    {
        s.pop();
    }
}
```

```
37      // This Method Clear the stack contents
38      //*********************************************************************
39      void clearStack(stack<string> &s)
40      {
41          while (!s.empty())
42          {
43              s.pop();
44          }
45      }
46
47      //*********************************************************************
48      // Method for navigation on key press in a Normal mode
49      //*********************************************************************
50      void navigate()
51      {
52
53          curPath = root;
54          xcor = 1, ycor = 80;
55          pos();
56          char ch;
57
58          struct termios initialrsettings, newrsettings;
59          tcgetattr(fileno(stdin), &initialrsettings);
60          // switch to canonical mode and echo mode
61          newrsettings = initialrsettings;
62          newrsettings.c_lflag &= ~ICANON;
63          newrsettings.c_lflag &= ~ECHO;
64
65          if (tcsetattr(fileno(stdin), TCSAFLUSH, &newrsettings) != 0)
66          {
67              fprintf(stderr, "Could not set attributes\n");
68          }
69          else
70          {
71
```

```
1
2
3   //*****************************************************************
4   // Header file Included
5   //*****************************************************************
6   #include "myheader.h"
7
8   //*****************************************************************
9   // Global declaration
10  //*****************************************************************
11  unsigned int rowsize, colsize;
12
13  //*****************************************************************
14  // Method returns count of maximun num of file to be printed on terminal
15  //*****************************************************************
16  int getFilePrintingcount()
17  {
18      int lenRecord;
19      struct winsize win;
20      ioctl(STDOUT_FILENO, TIOCGWINSZ, &win);
21      rowsize = win.ws_row - 1;
22      colsize = win.ws_col;
23      if (totalFiles <= rowsize)
24      {
25          lenRecord = totalFiles;
26      }
27      else
28      {
29          lenRecord = rowsize;
30      }
31      return lenRecord;
32  }
```