CAPSTONE PROJECT REPORT

Title: LINUX FILE EXPLORER

Student Name: Smruti Ranjan Bhuyan

Department: Computer Science & Information Technology

Tools & Technologies Used: C++, NCURSES Library, Linux Terminal, GCC Compiler, Makefile, Git & GitHub

Abstract

The Linux File Explorer project is a terminal-based file management system built using C++ and the ncurses library. It provides an interactive command-line interface that allows users to navigate directories, view file details, and perform basic file operations. The project aims to simulate a minimalistic yet efficient alternative to GUI-based file explorers, focusing on performance, keyboard control, and system-level understanding.

Objective

- To build a terminal-based file management system using C++.
- To display directory contents with file names, permissions, sizes, and timestamps.
- To implement keyboard-based navigation for user interaction.
- To understand low-level file handling and UI rendering using ncurses.

Problem Statement

Traditional GUI file managers, though user-friendly, may not be efficient in low-resource environments or remote servers. Hence, a terminal-based file explorer can serve as a lightweight alternative, providing faster operations and better resource usage. This project addresses the need for a structured and interactive file management interface in the terminal.

Existing System

Existing Linux file explorers like nautilus or thunar are GUI-based and resource-heavy. Command-line tools like ls and cd are non-interactive and lack a user interface for navigation.

Proposed System

The proposed Linux File Explorer combines the advantages of both: Lightweight terminal interface, Keyboard navigation, Dynamic file listing with attributes, and File operations like Open, Copy, Rename, Delete.

Novelty (Own Contribution)

Yes, novelty has been added to the existing concept. Beyond simple file listing, the project introduces display of file permissions and sizes, color-coded directory and file view, interactive header with author name and styling, dynamic scrolling and highlighting, clean and structured UI using ncurses, and keyboard-driven file navigation.

System Design

Architecture:

UIManager → Handles UI rendering using ncurses.
FileManager → Manages file operations.
main.cpp → Entry point controlling initialization and user interactions.

Flow:
Start → Initialize UI → Read Directory → Display Files → User Input → Action → Refresh → Exit

Modules

- UI Module: Displays file list and highlights.
- File Management Module: Reads directories and file metadata.
- Input Handling Module: Detects keyboard actions.
- Utility Module: Helper functions for paths and permissions.

Technologies Used

Component	Technology
Language	C++
UI Library	ncurses
Compiler	GCC
Build System	Makefile
Version Control	Git, GitHub

OS	Linux / Ubuntu

Implementation Details

Key Features: Displays files, shows permissions, highlights selection, and allows navigation through directories using keys. Modular structure ensures maintainability.

Full Source Code

main.cpp

```
#include "FileManager.h"
#include "UIManager.h"
#include <ncurses.h>

int main()
{
    initscr();
    start_color();
    use_default_colors();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);
    curs_set(0);

    // Define colors
    init_pair(1, COLOR_BLACK, COLOR_CYAN); // Header/Footer
    init_pair(2, COLOR_YELLOW, -1);       // Highlight
    init_pair(3, COLOR_WHITE, -1);        // Normal
    init_pair(4, COLOR_GREEN, -1);        // Success
    init_pair(5, COLOR_RED, -1);          // Error
    init_pair(6, COLOR_CYAN, -1);         // Help text
    init_pair(7, COLOR_BLUE, -1);         // Folder name

    FileManager fm;
    UIManager ui(fm);
    ui.start();

    endwin();
    return 0;
}
```

UIManager.h

```
#ifndef UIMANAGER_H

#define UIMANAGER_H

#include "FileManager.h"

#include <string>

#include <vector>

// A simple UI manager for console-based file explorer

class UIManager

{

private:

   FileManager &fm;

   int highlight;

   int offset;

   void drawHeader();

   void drawFooter(const std::vector<std::string> &files);

   void drawInstructions();

   void displayFiles(const std::vector<std::string> &files);

   void showHelp();

   void showShellPrompt();

   std::string prompt(const std::string &label);

public:

   UIManager(FileManager &fileManager);

   void start();

};

#endif
```

UIManager.cpp

```
#include "UIManager.h"

#include <ncurses.h>

#include <iomanip>

#include <sstream>

#include <cstdlib>

UIManager::UIManager(FileManager &fileManager)

    : fm(fileManager), highlight(0), offset(0) {}

 // Draw header

void UIManager::drawHeader()

{

    attron(COLOR_PAIR(1) | A_BOLD);

    mvprintw(0, 0, "  LINUX FILE EXPLORER Author: SMRUTI RANJAN BHUYAN  ");

    int cols = getmaxx(stdscr);

    for (int i = 22; i < cols; i++)

        printw(" ");

    attroff(COLOR_PAIR(1) | A_BOLD);

}

// Draw footer with current path and prompt

void UIManager::drawFooter(const std::vector<std::string> &files)

{

    int rows = getmaxy(stdscr);

    attron(COLOR_PAIR(1));

    unsigned long long freeGB = fm.getFreeSpace();

    std::ostringstream ss;

    ss << fm.getCurrentPath() << ":$ ";
    mvprintw(rows - 3, 0, " %-*s", getmaxx(stdscr) - 1, ss.str().c_str());
    attroff(COLOR_PAIR(1));
}
// Draw instructions at the bottom
void UIManager::drawInstructions()
{
    int rows = getmaxy(stdscr);
    attron(COLOR_PAIR(6));
    mvprintw(rows - 2, 0,
             "[Up|Down] Move  [Enter] Open  [Backspace] Up  "
             "[c] touch  [C] mkdir  [d] rm  [y] cp  [m] mv  "
             "[s] find  [p] chmod  [h] Help  [q] Quit");
    attroff(COLOR_PAIR(6));
}
// Prompt for user input
std::string UIManager::prompt(const std::string &label)
{
    echo();
    curs_set(1);
    char input[256];
    int rows = getmaxy(stdscr);
    mvprintw(rows - 1, 0, "%s: ", label.c_str());
    clrtoeol();
    getnstr(input, 255);
    noecho();
    curs_set(0);
    return std::string(input);
}
// Display files and directories
void UIManager::displayFiles(const std::vector<std::string> &files)
{
    clear();
    drawHeader();
    mvprintw(2, 0, "[DIR] Path: %s", fm.getCurrentPath().c_str());
    int rows, cols;
    getmaxyx(stdscr, rows, cols);
    int visible = rows - 8;
    if (highlight < offset)
        offset = highlight;
    else if (highlight >= offset + visible)
        offset = highlight - visible + 1;
    attron(A_BOLD | COLOR_PAIR(6));
    mvprintw(3, 2, "%-6s %-40s %-10s %-12s", "TYPE", "NAME", "SIZE", "PERMS");
    attroff(A_BOLD | COLOR_PAIR(6));
    for (int i = 0; i < visible && (i + offset) < (int)files.size(); ++i)
    {
        int index = i + offset;
        bool isDir = fm.isDirectory(files[index]);

        std::string perms = fm.getPermissions(files[index]);
        unsigned long long size = 0;
        if (!isDir)
            size = fm.getFileSize(files[index]);
        std::ostringstream sizeStr;
        if (isDir)
            sizeStr << "--";
        else if (size < 1024)
            sizeStr << size << " B";
        else if (size < 1024 * 1024)
            sizeStr << std::fixed << std::setprecision(1) << (size / 1024.0) << " KB";
        else
            sizeStr << std::fixed << std::setprecision(1) << (size / (1024.0 * 1024.0)) << " MB";
        if (isDir)
            attron(COLOR_PAIR(7) | A_BOLD);
        if (index == highlight)
            attron(COLOR_PAIR(2) | A_BOLD);
        mvprintw(i + 4, 2, "%-6s %-40s %-10s %-12s",
                isDir ? "[DIR]" : "[FIL]",
                files[index].c_str(),
                sizeStr.str().c_str(),
                perms.c_str());


        if (index == highlight)
            attroff(COLOR_PAIR(2) | A_BOLD);
        if (isDir)
            attroff(COLOR_PAIR(7) | A_BOLD);
    }
```

```
    drawFooter(files);
    drawInstructions();
    refresh();
}
// Show help menu
void UIManager::showHelp()
{
    clear();
    attron(A_BOLD | COLOR_PAIR(6));
    mvprintw(1, 2, "HELP MENU - FILE EXPLORER COMMANDS");
    attroff(A_BOLD | COLOR_PAIR(6));

    const char *helpText[] = {
        "[Up / Down] - Move Selection",
        "    Use arrow keys to navigate through files and folders.",
        "",
        "[Enter] - Open Folder/File",
        "    Opens the selected folder and displays its contents.",
        "    Opens the selected file with the default editor.",
        "",
        "[Backspace] - Go Up",
        "    Move one directory level up (to the parent folder).",
        "",
        "[c] - Create File",
        "    Prompts you to enter a new file name in the current directory.",
        "    Example: entering 'notes.txt' will create that file here.",
        "",
        "[d] - Delete File / Folder",
        "    Deletes the selected item permanently.",
        "    Use with caution — there is no undo.",
        "",
        "[y] - Copy File",
        "    Copies the selected file to a new location or name.",
        "    Example: 'report.txt' → enter 'backup_report.txt' to copy in same folder.",
        "    You can also specify a full path, e.g. '../backup/report.txt'.",
        "",
        "[m] - Move / Rename File",
        "    Moves or renames the selected file/folder.",
        "    Example: rename 'old.txt' → 'new.txt', or move to '../docs/new.txt'.",
        "",
        "[s] - Search Files",
        "    Prompts for a keyword and lists all matching files/folders in current directory.",
        "    Press any key to return to the main view after seeing results.",
        "",
        "[p] - View / Edit Permissions",
        "    Displays current Unix-style permissions (e.g., rwxr-xr--).",
        "    Enter 'y' to modify them — you will then be prompted to input new ones.",
        "    Example: 'rwxr--r--' means owner can read/write/execute, group read-only, others read-only.",
        "",
        "[h] - Show Help",
        "    Displays this help menu.",
        "",
        "[q] - Quit",
        "    Exits the file explorer safely.",
    };

    int row = 3;
    for (int i = 0; i < (int)(sizeof(helpText) / sizeof(helpText[0])); ++i)
    {
        mvprintw(row++, 4, "%s", helpText[i]);
    }

    attron(COLOR_PAIR(6));
    mvprintw(row + 1, 2, "Press any key to return...");
    attroff(COLOR_PAIR(6));

    refresh();
    getch();
}
// Main UI loop
void UIManager::start()
{
    std::vector<std::string> files = fm.listFiles();
    int ch;

    while (true)
    {
```

```
displayFiles(files);
ch = getch();
switch (ch)
{
case KEY_UP:
   if (highlight > 0)
      highlight--;
   break;

case KEY_DOWN:
   if (highlight < (int)files.size() - 1)
      highlight++;
   break;
case 10:
{
   if (files.empty())
      break;
   std::string target = files[highlight];
   if (fm.isDirectory(target))
   {
      fm.changeDirectory(target);
      files = fm.listFiles();
      highlight = 0;
      offset = 0;
   }
   else
```

```
            {

                std::string command = "start \"\" \"" + fm.getCurrentPath() + "/" + target + "\"";
                system(command.c_str());
                mvprintw(getmaxy(stdscr) - 1, 0, "[INFO] Opening '%s' with default editor.",
target.c_str());
            }
            break;
        }
        case KEY_LEFT:
        case KEY_BACKSPACE:
        case 127:
            fm.changeDirectory("..");
            files = fm.listFiles();
            highlight = 0;
            offset = 0;
            break;
        case 'c':
        {
            std::string name = prompt("touch");
            if (name.empty())
                break;
            bool ok = fm.createFile(name);
            mvprintw(getmaxy(stdscr) - 1, 0, ok ? "[OK] File created." : "[ERR] Failed to create
file.");
            files = fm.listFiles();
            break;
```

```
    }
    case 'C':
    {
        std::string name = prompt("mkdir");
        if (name.empty())
            break;
        bool ok = fm.createDirectory(name);
        mvprintw(getmaxy(stdscr) - 1, 0, ok ? "[OK] Directory created." : "[ERR] Failed to
create directory.");
        files = fm.listFiles();
        break;
    }
    case 'd':
    {
        if (files.empty())
            break;
        std::string name = files[highlight];
        bool ok = fm.deleteFile(name);
        mvprintw(getmaxy(stdscr) - 1, 0, ok ? "[OK] File deleted." : "[ERR] Delete failed.");
        files = fm.listFiles();
        if (highlight > 0)
            highlight--;
        break;
    }


    case 'y':
```

```
{
    if (files.empty())
        break;
    std::string src = files[highlight];
    std::string dest = prompt("cp");
    if (dest.empty())
        break;
    bool ok = fm.copyFile(src, dest);
    mvprintw(getmaxy(stdscr) - 1, 0, ok ? "[OK] File copied." : "[ERR] Copy failed.");
    files = fm.listFiles();
    break;
}
case 'm':
{
    if (files.empty())
        break;
    std::string src = files[highlight];
    std::string dest = prompt("mv");
    if (dest.empty())
        break;
    bool ok = fm.moveFile(src, dest);
    mvprintw(getmaxy(stdscr) - 1, 0, ok ? "[OK] File moved." : "[ERR] Move failed.");
    files = fm.listFiles();
    break;
}
```

```
case 's':
{
    std::string key = prompt("find");
    if (key.empty())
        break;
    auto results = fm.searchFiles(key);
    clear();
    mvprintw(1, 2, "Search results for '%s':", key.c_str());
    int i = 3;
    for (auto &r : results)
    {
        mvprintw(i++, 4, "%s", r.c_str());
        if (i >= getmaxy(stdscr) - 2)
            break;
    }
    mvprintw(getmaxy(stdscr) - 1, 2, "Press any key to return...");
    refresh();
    getch();
    files = fm.listFiles();
    break;
}

case 'p':
{
    if (files.empty())
        break;
    std::string name = files[highlight];
    std::string perms = fm.getPermissions(name);
    mvprintw(getmaxy(stdscr) - 3, 0, "%s : %s", name.c_str(), perms.c_str());
    refresh();

    std::string choice = prompt("chmod");
    if (choice.empty())
        break;
    if (choice.size() == 9)
    {
        bool ok = fm.setPermissions(name, choice);
        mvprintw(getmaxy(stdscr) - 1, 0,
                 ok ? "[OK] Permissions updated." : "[ERR] Failed to update perms.");
    }
    else
    {
        mvprintw(getmaxy(stdscr) - 1, 0, "[ERR] Invalid format (use rwxrwxrwx).");
    }
    break;
}

case 'h':
    showHelp();
    break;

case 'q':
```

```
            return;
        }
    }
}
```

FileManager.h

```
#ifndef FILEMANAGER_H

#define FILEMANAGER_H

#include <string>

#include <vector>


class FileManager {

private:

    std::string currentPath;

public:

    FileManager();

    std::vector<std::string> listFiles() const;

    bool changeDirectory(const std::string &dir);

    std::string getCurrentPath() const;

    bool createFile(const std::string &filename);

    bool createDirectory(const std::string &dirname);

    bool deleteFile(const std::string &filename);

    bool copyFile(const std::string &src, const std::string &dest);

    bool moveFile(const std::string &src, const std::string &dest);

    unsigned long long getFileSize(const std::string &name);
    std::vector<std::string> searchFiles(const std::string &keyword) const;

    std::string getPermissions(const std::string &filename) const;

    bool setPermissions(const std::string &filename, const std::string &mode);

    bool isDirectory(const std::string &name) const;

    unsigned long long getFreeSpace() const;

};

#endif
```

FileManager.cpp

```
#include "FileManager.h"

#include <filesystem>

#include <fstream>

#include <sys/stat.h>


namespace fs = std::filesystem;

// Constructor for FileManager

FileManager::FileManager() {

    currentPath = fs::current_path().string();

}

// List files in the current directory

std::vector<std::string> FileManager::listFiles() const {

    std::vector<std::string> files;

    try {

        for (const auto &entry : fs::directory_iterator(currentPath))

            files.push_back(entry.path().filename().string());
    } catch (...) {}
    return files;
}

// Change the current directory
bool FileManager::changeDirectory(const std::string &dir) {
    fs::path newPath = (dir == "..") ? fs::path(currentPath).parent_path()
                                     : fs::path(currentPath) / dir;
    if (fs::exists(newPath) && fs::is_directory(newPath)) {
        currentPath = fs::canonical(newPath).string();
        return true;
    }
    return false;
}

// Get the current directory path
std::string FileManager::getCurrentPath() const {
    return currentPath;
}

bool FileManager::createFile(const std::string &filename) {
    try {
        std::ofstream file((fs::path(currentPath) / filename).string());
        return file.good();
    } catch (...) { return false; }
}
```

```
// Create a new directory
bool FileManager::createDirectory(const std::string &dirname) {
   try {
      return fs::create_directory(fs::path(currentPath) / dirname);
   } catch (...) { return false; }
}

// Delete a file
bool FileManager::deleteFile(const std::string &filename) {
   try { return fs::remove(fs::path(currentPath) / filename); }
   catch (...) { return false; }
}

// Copy a file
bool FileManager::copyFile(const std::string &src, const std::string &dest) {
   try {
      fs::copy(fs::path(currentPath) / src, fs::path(currentPath) / dest,
               fs::copy_options::overwrite_existing);
      return true;
   } catch (...) { return false; }
}

// Move a file
bool FileManager::moveFile(const std::string &src, const std::string &dest) {
   try {
        fs::rename(fs::path(currentPath) / src, fs::path(currentPath) / dest);
        return true;
    } catch (...) { return false; }
}

// Search files by keyword
std::vector<std::string> FileManager::searchFiles(const std::string &keyword) const {
    std::vector<std::string> results;
    try {
        for (auto &entry : fs::recursive_directory_iterator(currentPath)) {
            if (entry.path().filename().string().find(keyword) != std::string::npos)
                results.push_back(entry.path().string());
        }
    } catch (...) {}
    return results;
}

// Get file permissions
std::string FileManager::getPermissions(const std::string &filename) const {
    struct stat info;
    std::string perms = "---------";
    if (stat((fs::path(currentPath) / filename).c_str(), &info) == 0) {
        perms[0] = (info.st_mode & S_IRUSR) ? 'r' : '-';
        perms[1] = (info.st_mode & S_IWUSR) ? 'w' : '-';
        perms[2] = (info.st_mode & S_IXUSR) ? 'x' : '-';
        perms[3] = (info.st_mode & S_IRGRP) ? 'r' : '-';
        perms[4] = (info.st_mode & S_IWGRP) ? 'w' : '-';
        perms[5] = (info.st_mode & S_IXGRP) ? 'x' : '-';
        perms[6] = (info.st_mode & S_IROTH) ? 'r' : '-';
        perms[7] = (info.st_mode & S_IWOTH) ? 'w' : '-';
        perms[8] = (info.st_mode & S_IXOTH) ? 'x' : '-';
    }
    return perms;
}
// Set file permissions
bool FileManager::setPermissions(const std::string &filename, const std::string &mode) {
    fs::path filePath = fs::path(currentPath) / filename;
    try {
        fs::perms newPerms = fs::perms::none;
        if (mode.size() == 9) {
            if (mode[0] == 'r') newPerms |= fs::perms::owner_read;
            if (mode[1] == 'w') newPerms |= fs::perms::owner_write;
            if (mode[2] == 'x') newPerms |= fs::perms::owner_exec;
            if (mode[3] == 'r') newPerms |= fs::perms::group_read;
            if (mode[4] == 'w') newPerms |= fs::perms::group_write;
            if (mode[5] == 'x') newPerms |= fs::perms::group_exec;
            if (mode[6] == 'r') newPerms |= fs::perms::others_read;
            if (mode[7] == 'w') newPerms |= fs::perms::others_write;
            if (mode[8] == 'x') newPerms |= fs::perms::others_exec;
        }
        fs::permissions(filePath, newPerms, fs::perm_options::replace);
        return true;
    } catch (...) {
        return false;
    }
}
// Check if a path is a directory
bool FileManager::isDirectory(const std::string &name) const {
    return fs::is_directory(fs::path(currentPath) / name);
}
// Get free space in the current directory (in GB)
unsigned long long FileManager::getFreeSpace() const {
    try {
        auto space = fs::space(currentPath);
        return space.available / (1024 * 1024 * 1024);
    } catch (...) { return 0; }
}


// Get file size
unsigned long long FileManager::getFileSize(const std::string &name)
{
    std::string fullPath = currentPath + "/" + name;
    struct stat st;
    if (stat(fullPath.c_str(), &st) == 0)
        return st.st_size;
    return 0;
}
```

Result

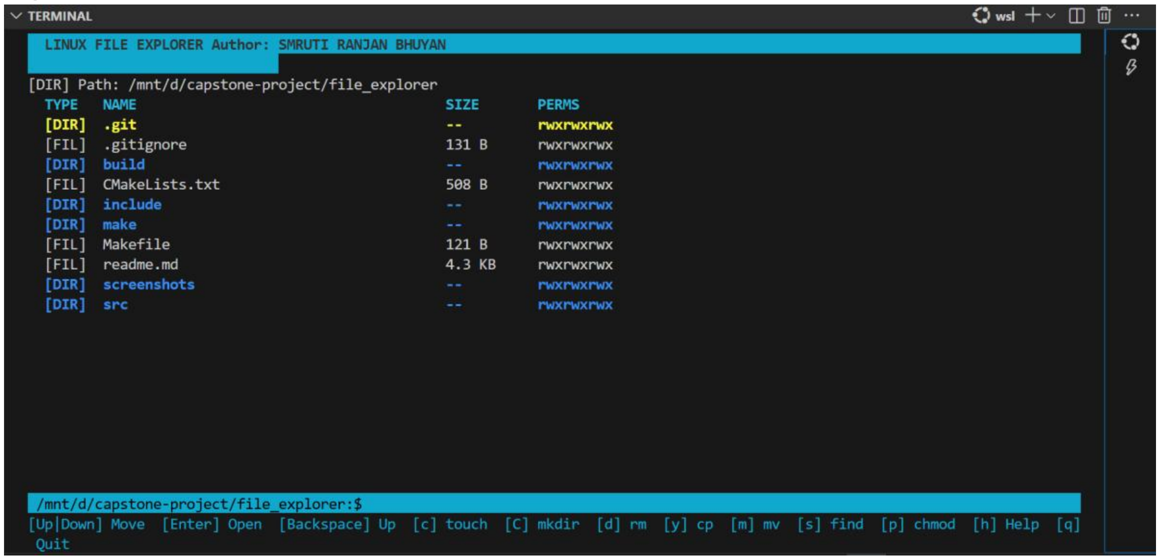
The project successfully provides a terminal-based interactive file explorer. It achieves the goal of building a lightweight, efficient, and educational tool.

Conclusion

The project demonstrates how C++ and ncurses can be used to build powerful terminal applications. Future improvements could include mouse support, file preview, and sorting options.

Screenshots

Figure 1: Header View

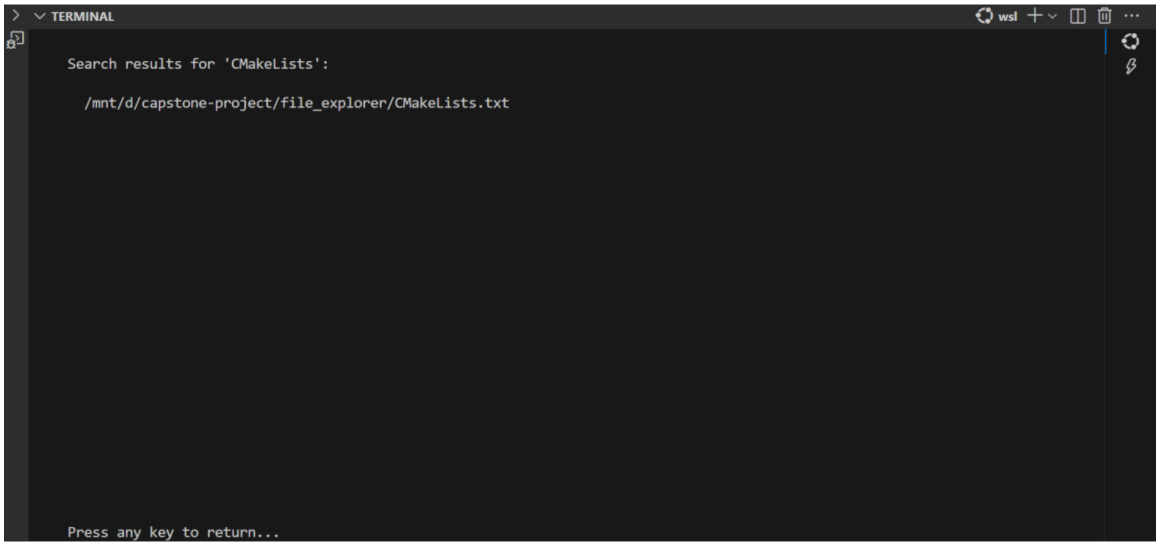


```
 LINUX FILE EXPLORER Author: SMRUTI RANJAN BHUYAN

[DIR] Path: /mnt/d/capstone-project/file_explorer
  TYPE   NAME                                         SIZE        PERMS
  [DIR]  .git                                         --          rwxrwxrwx
  [FIL]  .gitignore                                   131 B       rwxrwxrwx
  [DIR]  build                                        --          rwxrwxrwx
  [FIL]  CMakeLists.txt                               508 B       rwxrwxrwx
  [DIR]  include                                      --          rwxrwxrwx
  [DIR]  make                                         --          rwxrwxrwx
  [FIL]  Makefile                                     121 B       rwxrwxrwx
  [FIL]  readme.md                                    4.3 KB      rwxrwxrwx
  [DIR]  screenshots                                  --          rwxrwxrwx
  [DIR]  src                                          --          rwxrwxrwx

 /mnt/d/capstone-project/file_explorer:$
[Up|Down] Move  [Enter] Open  [Backspace] Up  [c] touch  [C] mkdir  [d] rm  [y] cp  [m] mv  [s] find  [p] chmod  [h] Help  [q]
 Quit
```

Figure 2: File Search View

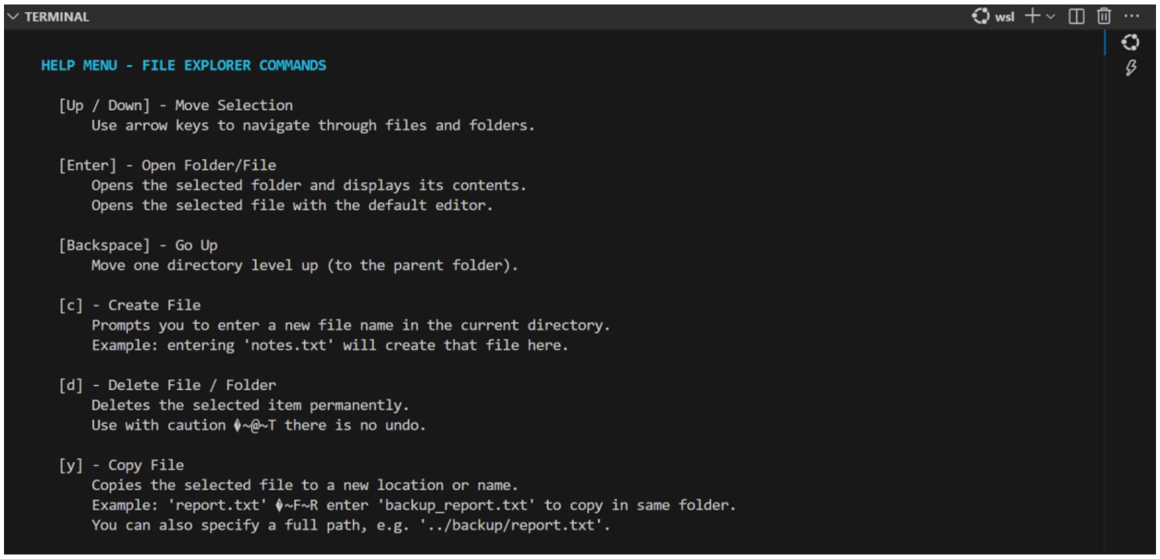


```
    Search results for 'CMakeLists':

      /mnt/d/capstone-project/file_explorer/CMakeLists.txt


    Press any key to return...
```

Figure 3: Help Menu



```
HELP MENU - FILE EXPLORER COMMANDS

  [Up / Down] - Move Selection
      Use arrow keys to navigate through files and folders.

  [Enter] - Open Folder/File
      Opens the selected folder and displays its contents.
      Opens the selected file with the default editor.

  [Backspace] - Go Up
      Move one directory level up (to the parent folder).

  [c] - Create File
      Prompts you to enter a new file name in the current directory.
      Example: entering 'notes.txt' will create that file here.

  [d] - Delete File / Folder
      Deletes the selected item permanently.
      Use with caution ⧫~@~T there is no undo.

  [y] - Copy File
      Copies the selected file to a new location or name.
      Example: 'report.txt' ⧫~F~R enter 'backup_report.txt' to copy in same folder.
      You can also specify a full path, e.g. '../backup/report.txt'.
```