INTERACTIVE FILE MANAGEMENT SYSTEM

Simulating Core File Operations

Presentation By:

Sai Kiran **Harsh Patel**

Kaustubh Adhav **Dhruvik Patel**

OVERVIEW

This project implements a file system GUI using Dear ImGui, a graphical interface library. It enables users to perform file system operations including file and directory management, disk usage monitoring, and file modification.

Implemented in C++ with OpenGL for rendering.

IMPLEMENTED FUNCTIONS

Create Directory	List Directory Contents
Delete Directory	Rename File/Directory
Create File	Move File/Directory
Delete File	Copy File
Write to File	Change File Permissions
Read File	Get Disk Usage
Get File Info	

LIBRARIES AND THEIR USES

- Standard C++ Libraries:
- <iostream>, <fstream>, <string>, <vector>, <filesystem> <sys/stat.h>, <sys/types.h>
- Dear ImGui: UI elements for file operations.
- GLFW: OpenGL context management and user input.
- OpenGL: Graphics rendering.
- POSIX APIs: OS-level file operations.

DIRECTORY STRUCTURE

- ├── FileSys_GUI
- ├── imgui
- ├── Output_Screenshots
- └── README.md

DETAILED FILE ANALYSIS

- file_operations.cpp: - Implements core file operations (e.g., create, delete, list, write).
- file_operations.h: - Function prototypes for file_operations.cpp.
- main.cpp: - GUI integration with file operations using Dear ImGui.
- Makefile: - Automates build process.

file_operations.cpp

Purpose:

- Implements the core file system operations, handling functionalities like creating, deleting, and managing files and directories.

Main Libraries Used:

- <filesystem>: For interacting with the file system.
- <sys/stat.h>: To manage file and directory attributes.

Key Functions:

- create_directory: Creates a new directory.
- delete_directory: Deletes an existing directory.
- list_directory_contents: Lists all files and subdirectories in a specified directory.
- create_file: Creates a new file in a directory.
- delete_file: Deletes a specified file.
- write_to_file: Writes data to a file.
- read_file: Reads content from a file.
- change_permissions: Updates file permissions.

Role in the Project:

- Acts as the backbone for all file system-related functionalities, enabling the GUI to execute the desired operations seamlessly.

file_operations.h

- Header file declaring functions for file and directory operations.
- Provides an interface to the main program to utilize these operations.

Purpose:

- Defines the prototypes for functions implemented in file_operations.cpp.

Included Libraries:

- While it doesn't directly use libraries, it includes essential standard libraries like <string> and <vector> for handling strings and dynamic collections.

Role in the Project:

- Serves as the header file that declares all file system operations such as creating, deleting, and managing files and directories, which are implemented in the corresponding .cpp file.

Create Directory

```cpp
int create_directory(const char *name) {
    if (mkdir(name, 0777) == -1) {
        perror("mkdir failed");
        return errno;
    } else {
        std::cout << "Directory created: " << name << std::endl;
    }
    return 0;
}
```

Create Directory

```
// Directory creation
ImGui::Text("Directory Name");
ImGui::InputText("##DirectoryName", dirName, IM_ARRAYSIZE(dirName));
if (ImGui::Button("Create Directory")) {
    result = create_directory(dirName);
    if (result == 0) {
        snprintf(statusMessage, IM_ARRAYSIZE(statusMessage), "Directory created successfully: %s", dirName);
    } else {
        snprintf(statusMessage, IM_ARRAYSIZE(statusMessage), "Error creating directory: %s (%s)", dirName, strerror(result));
    }
}
```

Rename File/Directory

```
int rename_file_or_directory(const char *old_name, const char *new_name) {
    if (rename(old_name, new_name) == -1) {
        perror("rename failed");
        return errno;
    } else {
        std::cout << "Renamed: " << old_name << " to " << new_name << std::endl;
    }
    return 0;
}
```

Change Permissions

```
int change_permissions(const char *path, mode_t mode) {
    if (chmod(path, mode) == -1) {
        perror("chmod failed");
        return errno;
    } else {
        std::cout << "Permissions changed for: " << path << std::endl;
    }
    return 0;
}
```

Disk Usage

```
struct statvfs  get_disk_usage(const char* path) {
    struct statvfs stat;

    // Get filesystem stats
    if (statvfs(path, &stat) != 0) {
        perror("statvfs failed");
        return stat;
    }

    // You can now use the stat structure to get disk usage info
    unsigned long free_space = stat.f_bfree * stat.f_frsize;
    unsigned long total_space = stat.f_blocks * stat.f_frsize;
    unsigned long used_space = total_space - free_space;

    // std::cout << "Free space: " << free_space << " bytes\n";
    // std::cout << "Used space: " << used_space << " bytes\n";
    // std::cout << "Total space: " << total_space << " bytes\n";

    return stat;

}
```

List Directory

```
std::string list_directory_contents(const char *path) {
    DIR *dir = opendir(path);
    if (dir == NULL) {
        return "Error: " + std::string(strerror(errno));
    }

    struct dirent *entry;
    std::string contents;
    while ((entry = readdir(dir)) != NULL) {
        contents += entry->d_name;
        contents += "\n";
    }

    closedir(dir);
    return contents;
}
```

List Directory

```
// Display directory contents if available
ImGui::InputTextMultiline("##dirContents", &dirContents[0], dirContents.size() + 1, ImVec2(-FLT_MIN, ImGui::GetTextLineHeigh

// List directory contents
if (ImGui::Button("List Directory Contents")) {
    ImGui::Text("Directory contents for: %s", filePath);
    dirContents = list_directory_contents(filePath);  // Get contents of the specified directory
    if (dirContents.rfind("Error:", 0) == 0) {
        snprintf(statusMessage, IM_ARRAYSIZE(statusMessage), "%s", dirContents.c_str());
        dirContents.clear();  // Clear if it's an error message
    } else {
        snprintf(statusMessage, IM_ARRAYSIZE(statusMessage), "Directory contents for: %s", filePath);
    }
}
```

main.cpp

Sets up the graphical interface using Dear ImGui:

Includes libraries for GUI functionality (imgui.h, etc.)

Provides an interactive GUI for file system operations:

Creating files

Deleting files

Reading files

Listing directory contents

Updates status messages for user feedback.

makefile

Provides build instructions for cross-platform compilation:

Linux: Links GLFW and OpenGL using pkg-config

Defines rules for compiling .cpp files and producing the executable.

Steps To Build And Run The Program

1. Install necessary libraries:

- sudo apt update
- sudo apt install gcc pkg-config g++ build-essential libglfw3-dev libgl1-mesa-dev libx11-dev libxrandr-dev libxi-dev libxxf86vm-dev libxcursor-dev cmake

2. Clone Dear ImGui repository:

- git clone --recursive https://github.com/ocornut/imgui -b docking

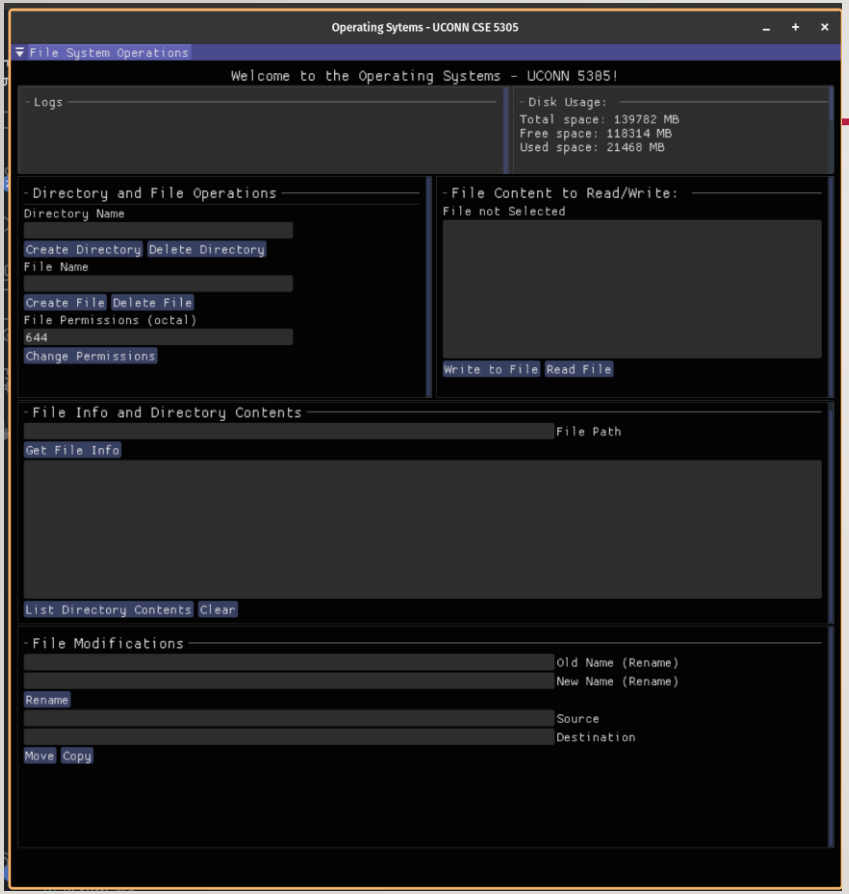
3. Build the project:

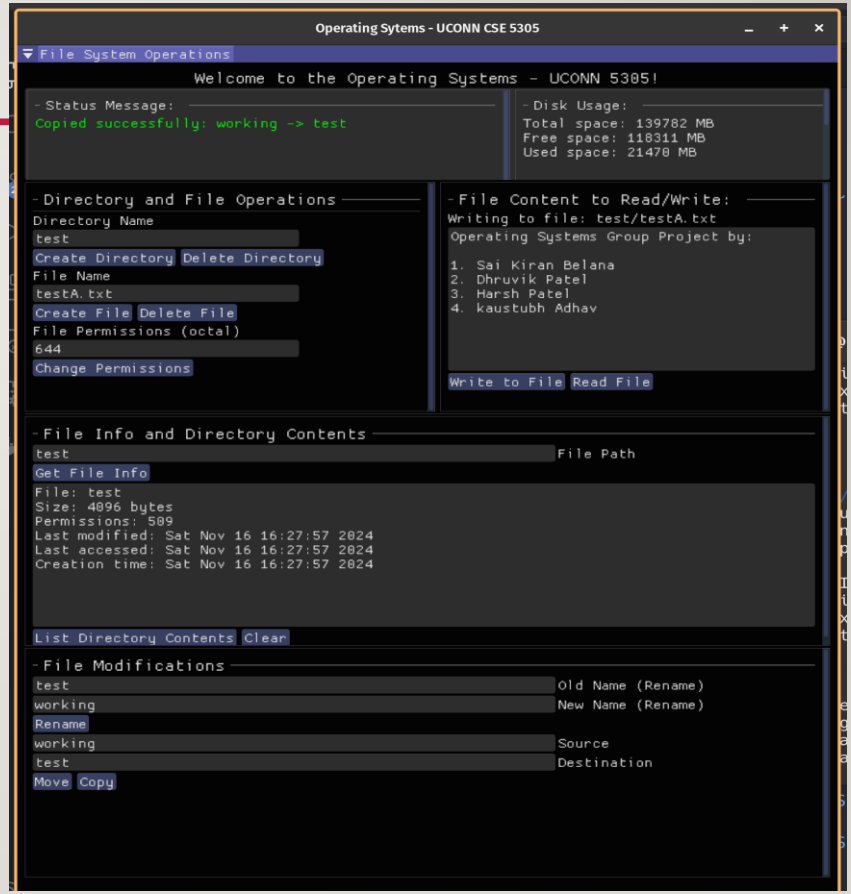
- cd file_system_gui
- make

4. Run the GUI interface:

- ./file_Sys_gui

Screenshots





Thank You