OS Group Project

Interactive File Management System

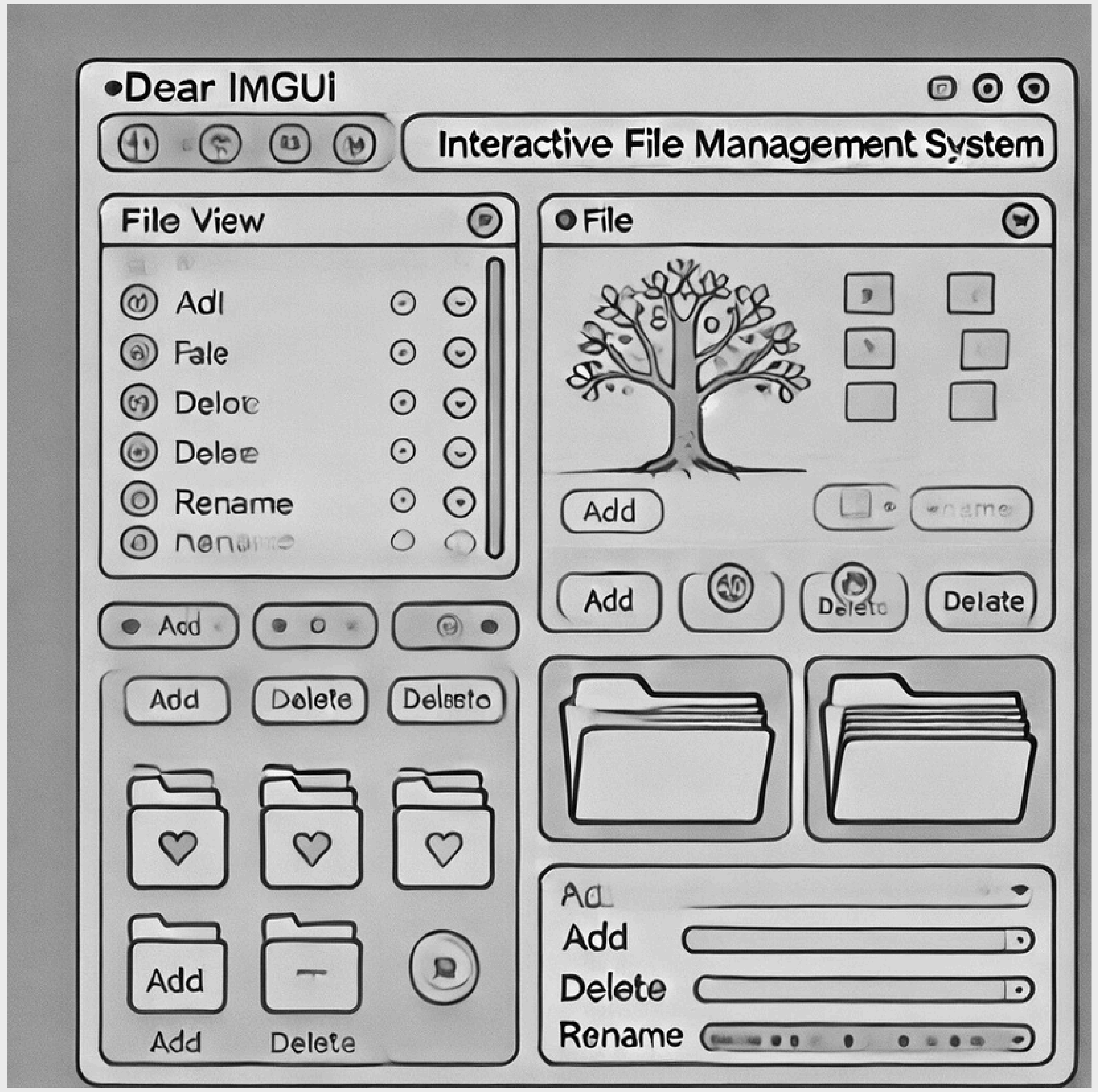
By,

Sai Kiran Belana
Dhruvik Patel
Harsh patel
Kaustubh Adhav

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.



Application Architecture

High-Level Design: The application consists of two main components:

1. **User Interface (UI)**: Built using Dear ImGui, it provides interactive elements for user input and displays information.
2. **Backend Logic:** Implements file system operations and interacts with the operating system to perform tasks requested through the UI.

Component Interaction: The UI captures user inputs and triggers corresponding backend functions to execute file system operations. The results are then displayed back on the UI.

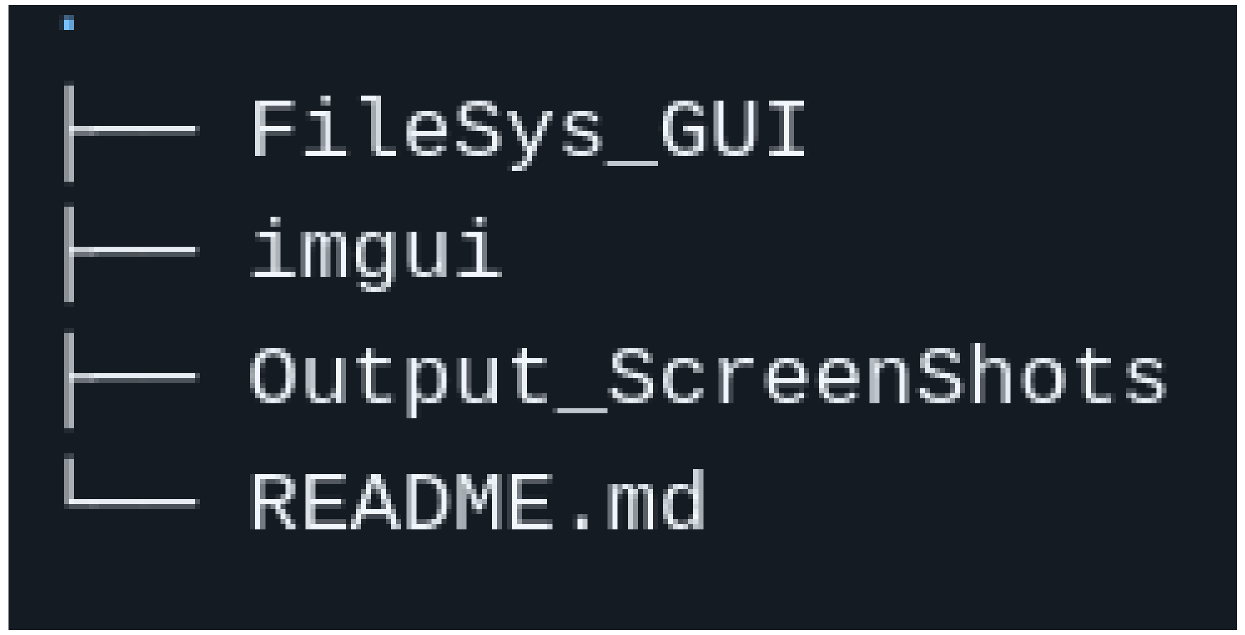
Core Functionalities Implemented:

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 Used:

- 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
```

File Analysis

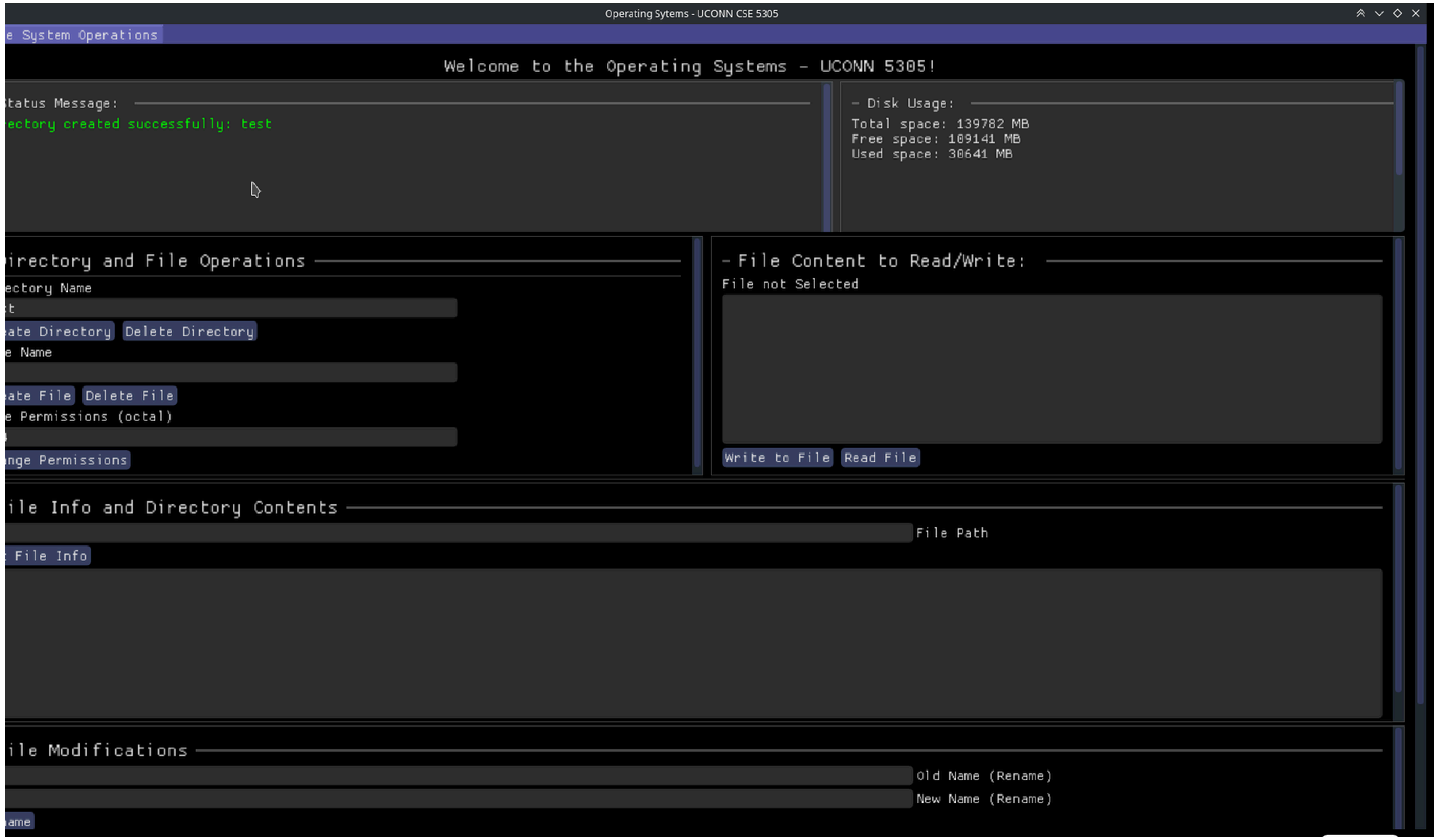
file_operations.cpp: Implements core file operations (e.g., create, delete, list, write).

file_operations.h: - Function prototypes for file_operations.cpp.

main.cpp: GUI integration with file operations using Dear ImGui.

Makefile: Automates build process.

Glimpse of GUI



BackEnd

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.
- rename_file: renames a given file
- move_file: moves from source to destination
- copy_file: copies source file to destination path
- get_file_info: displays the file information
- get_disk_usage: displays the total disk utilization

Role in the Project

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

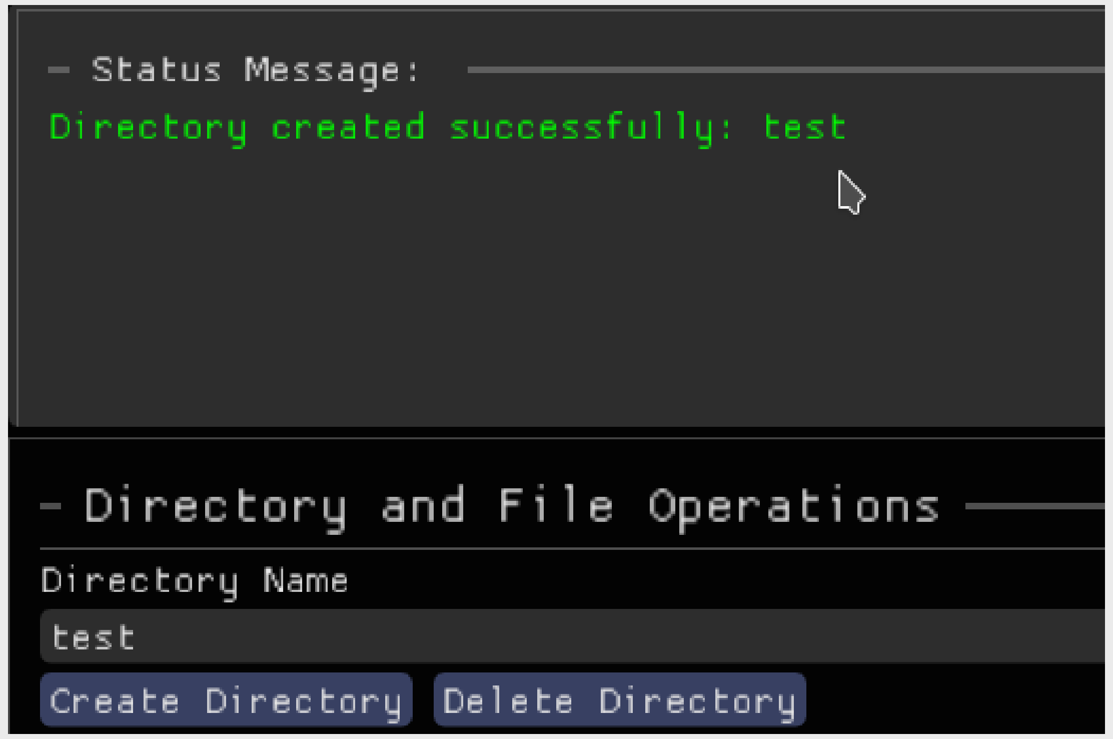
Code Walk through

file_operations.cpp

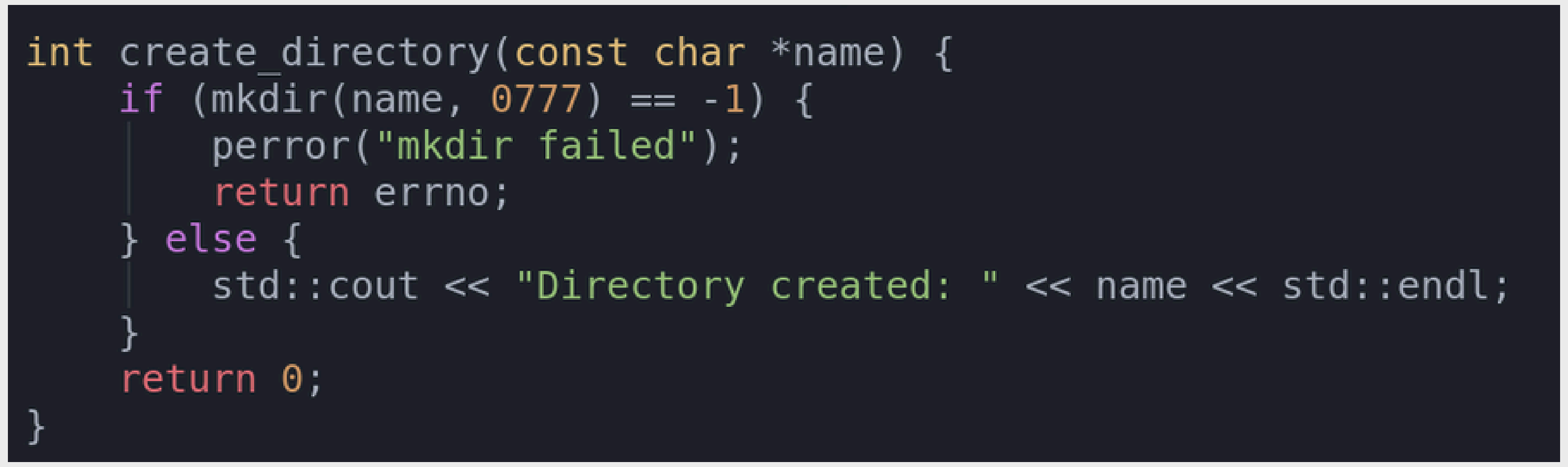
create directory:

Creates a new directory with default permissions (0777)

- Uses the mkdir system call to attempt directory creation.
- If successful:
  - Prints a success message in returns 0 handled by GUI
- If unsuccessful:
  - Uses perror to print the error message and returns 1 handles GUI error handled by main.cpp
  - Returns the errno value for further debugging.

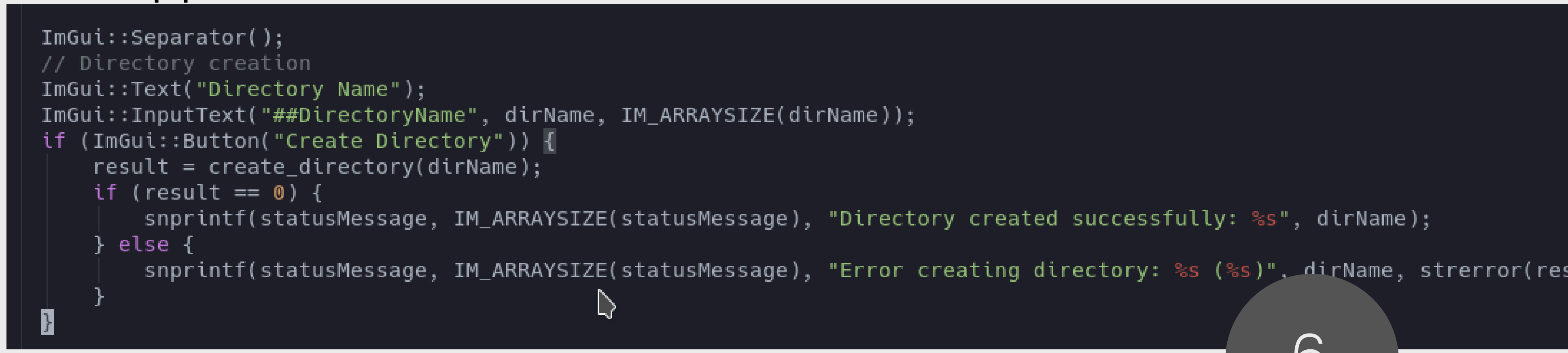


```
Status Message:
Directory created successfully: test
```



```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;
}
```

main.cpp - GUI button

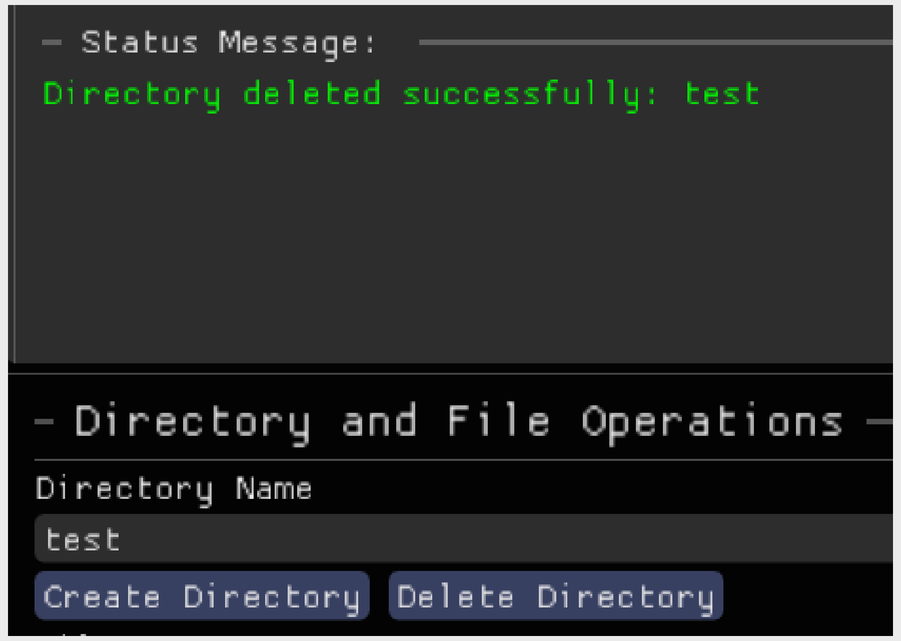


```cpp
ImGui::Separator();
// Directory creation
ImGui::Text("Directory Name");
ImGui::InputText("##DirectoryName", dirName, IM_ARRAYSIZE(dirName));
if (ImGui::Button("Create Directory")) {
    result = create_directory(dirName);
    if (result == 0) {
        snprintf(statusMessage, IM_ARRAYSIZE(statusMessage),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

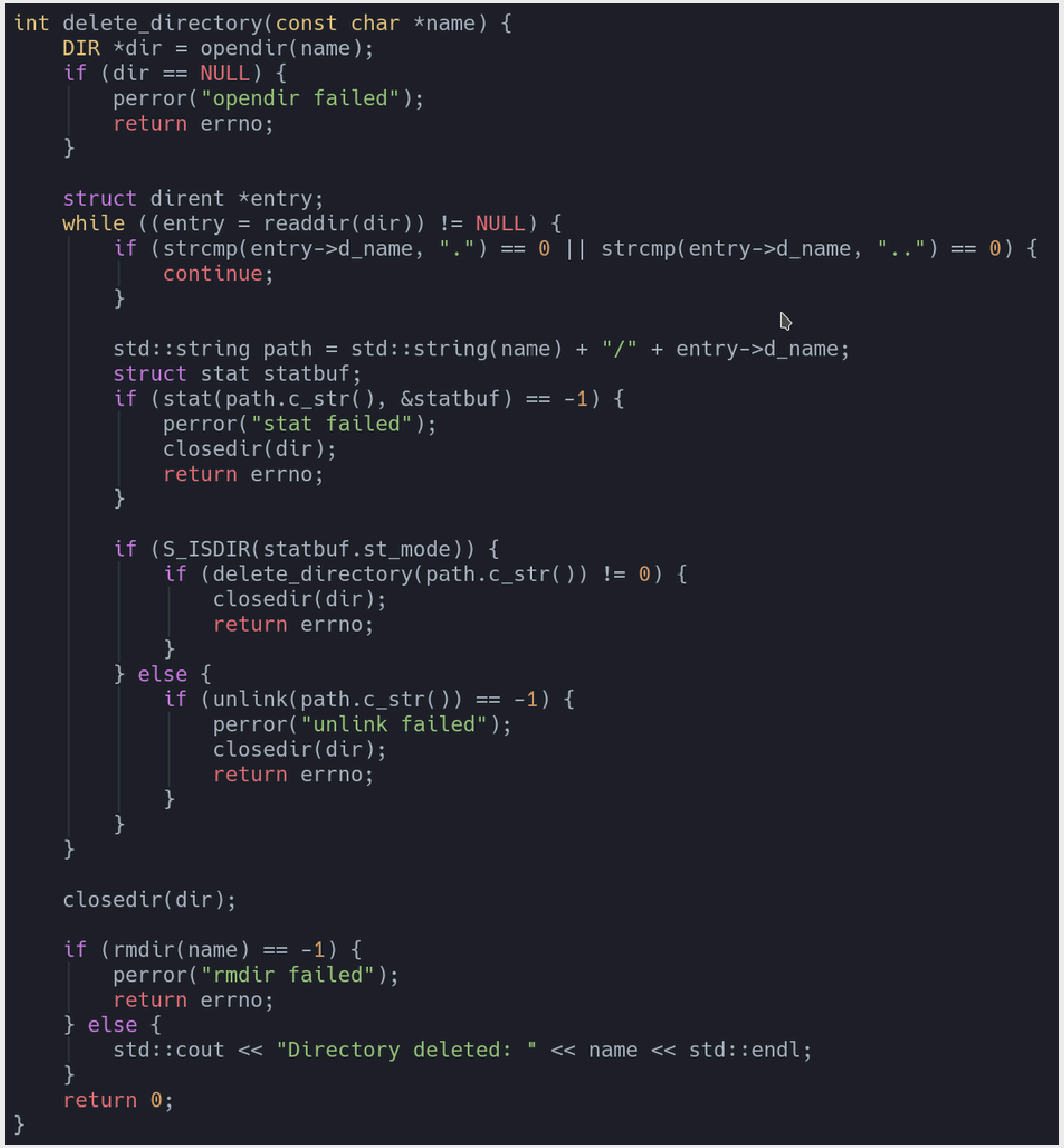
delete directory:

Deletes a directory and all its contents recursively.

1. Opens the directory using opendir.
2. Iterates over directory entries using readdir.
3. Skips special entries . and ..
4. For each entry:
    - Uses stat to check if it's a file or directory.
    - If a file, deletes it with unlink.
    - If a directory, recursively calls delete_directory.
5. Closes the directory with closedir and removes the directory with rmdir.



```
Status Message:
Directory deleted successfully: test
```



```cpp
int delete_directory(const char *name) {
    DIR *dir = opendir(name);
    if (dir == NULL) {
        perror("opendir failed");
        return errno;
    }

    struct dirent *entry;
    while ((entry = readdir(dir)) != NULL) {
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0) {
            continue;
        }

        std::string path = std::string(name) + "/" + entry->d_name;
        struct stat statbuf;
        if (stat(path.c_str(), &statbuf) == -1) {
            perror("stat failed");
            closedir(dir);
            return errno;
        }

        if (S_ISDIR(statbuf.st_mode)) {
            if (delete_directory(path.c_str()) != 0) {
                closedir(dir);
                return errno;
            }
        } else {
            if (unlink(path.c_str()) == -1) {
                perror("unlink failed");
                closedir(dir);
                return errno;
            }
        }
    }

    closedir(dir);

    if (rmdir(name) == -1) {
        perror("rmdir failed");
        return errno;
    } else {
        std::cout << "Directory deleted: " << name << std::endl;
    }
    return 0;
}
```

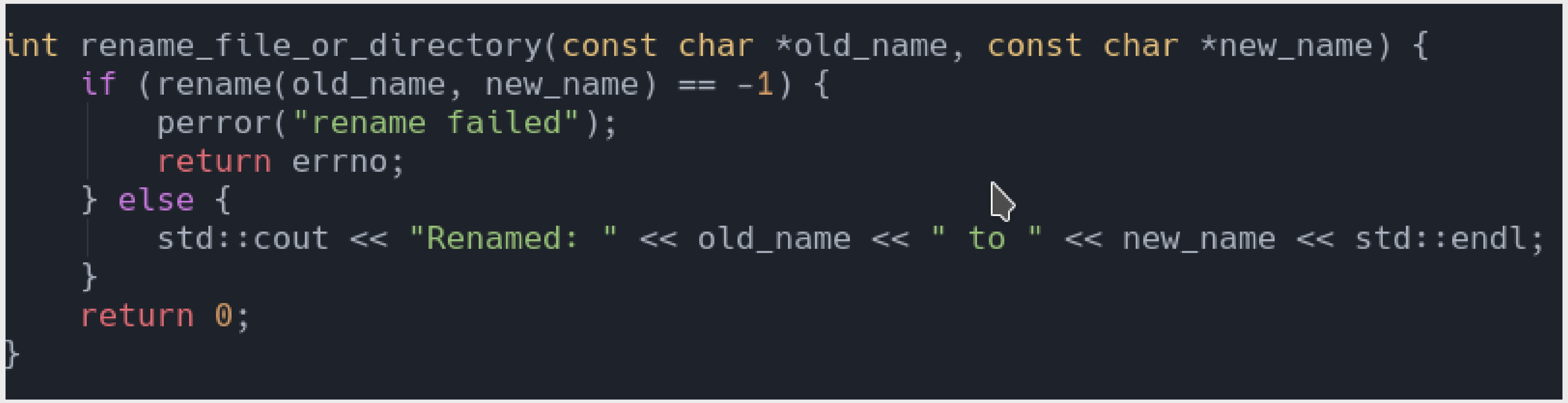
Renaming File/Directory

Renames a given directory/file and handles the error function.

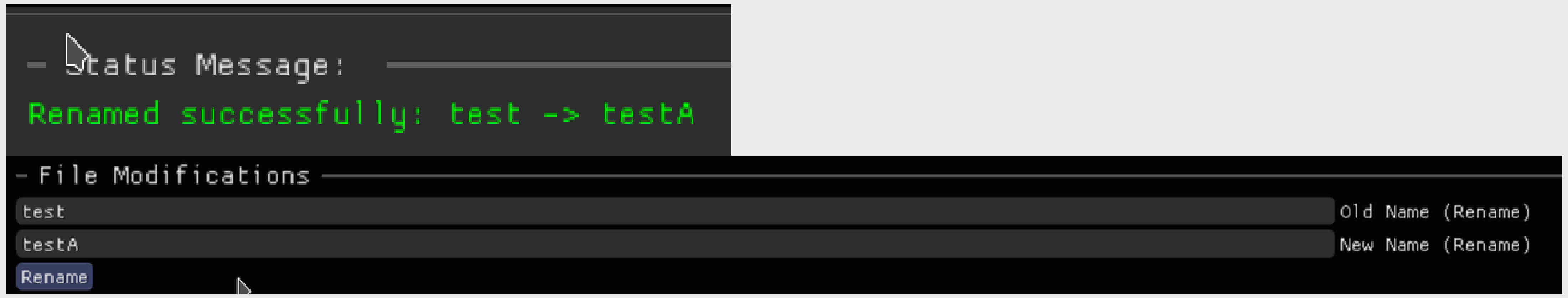
Accepts two arguments

- Old Name
- New Name

Using rename system call from library stdio



```
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;
}
```



```
Status Message:
Renamed successfully: test -> testA
```

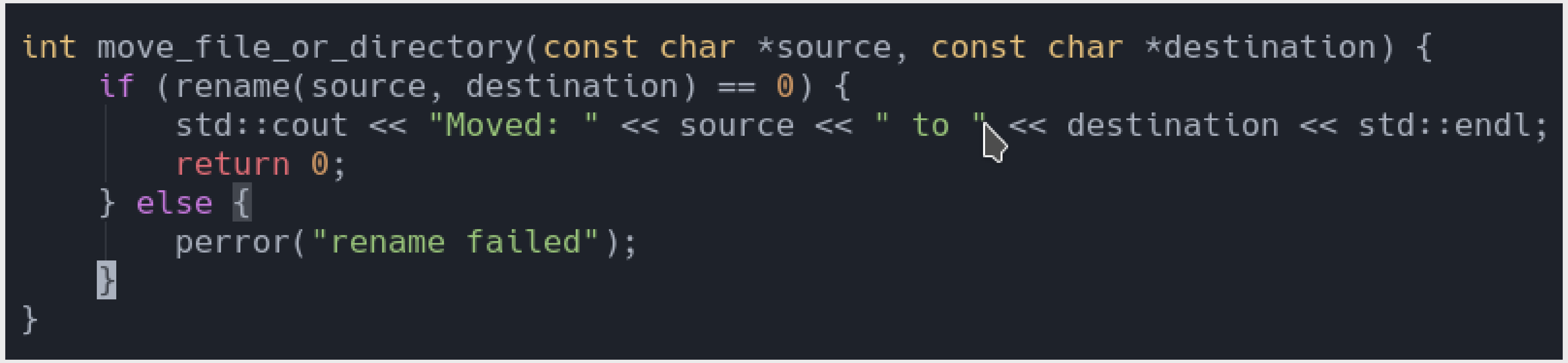
Moving Files/Directory

Moves file/dir from source to destination

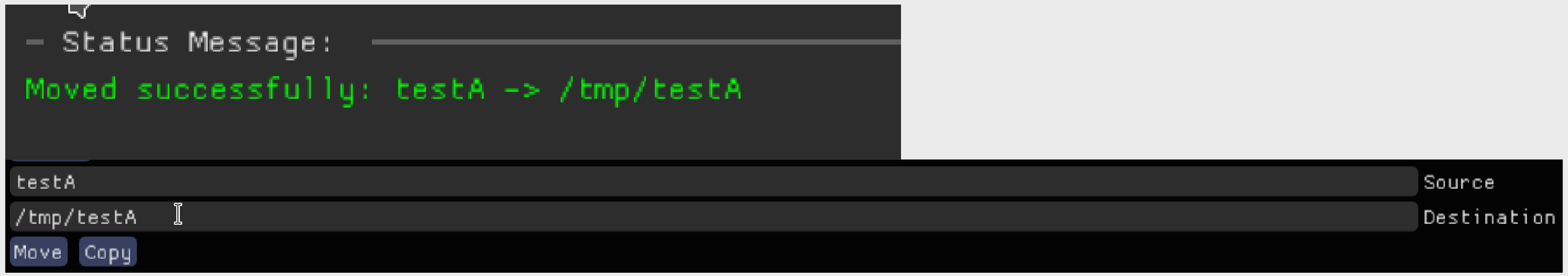
Accepts two arguments

- Source path
- Destination path

Uses rename for a fast move within the same filesystem.



```
int move_file_or_directory(const char *source, const char *destination) {
    if (rename(source, destination) == 0) {
        std::cout << "Moved: " << source << " to " << destination << std::endl;
        return 0;
    } else {
        perror("rename failed");
    }
}
```



```
— Status Message:
Moved successfully: testA -> /tmp/testA
```

```
testA                                   Source
/tmp/testA                              Destination
```

Copying Files/Directory

Copies a file to a new location.

Accepts two arguments

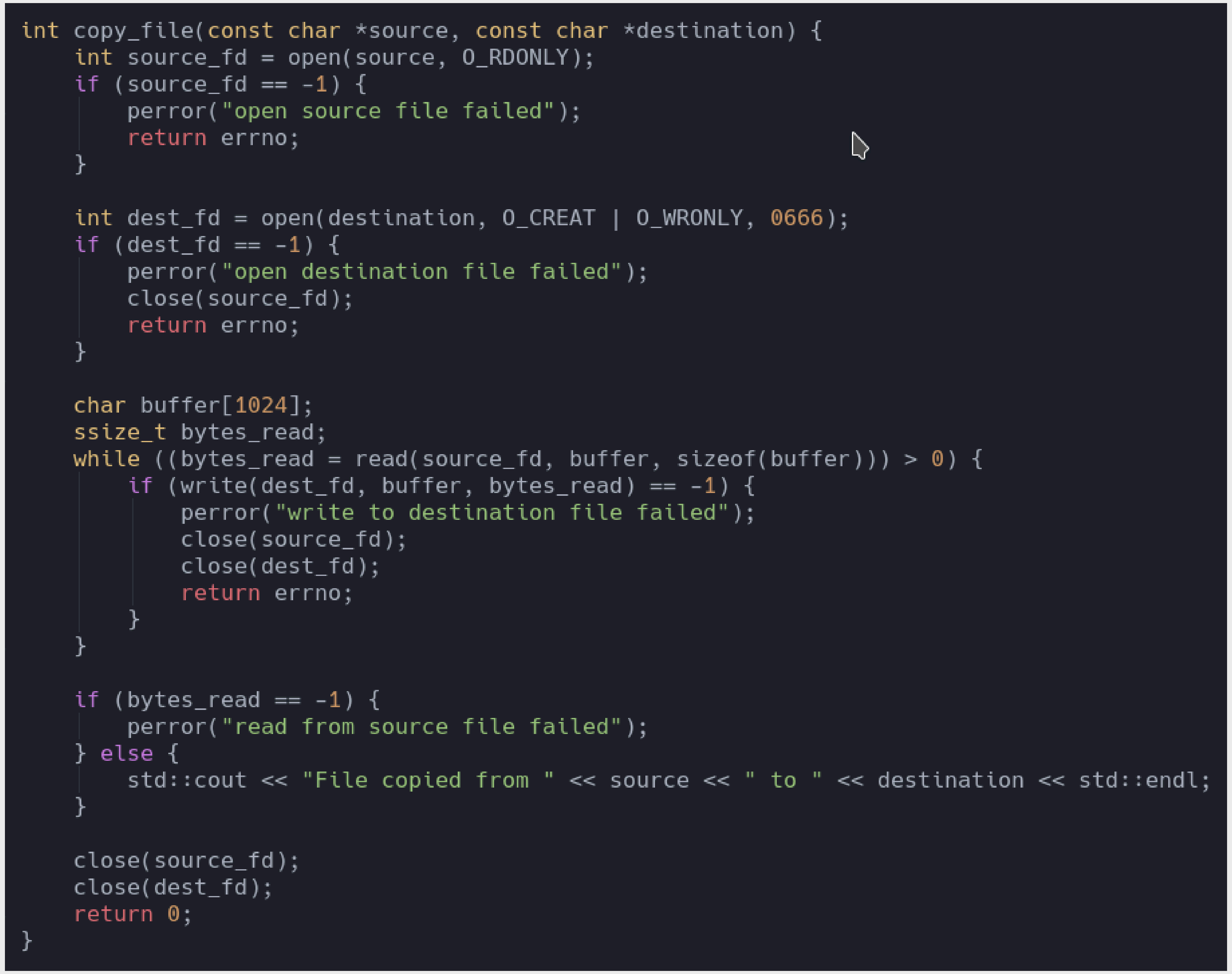
- Source path
- Destination path
-

Implementation:

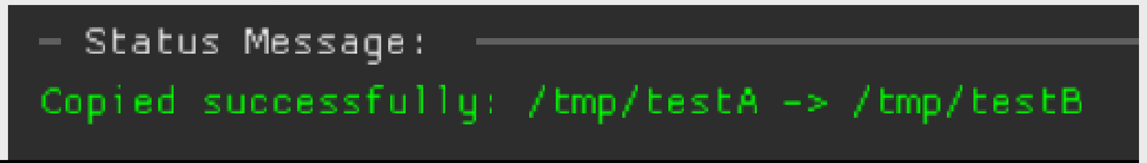
1. Opens the source and destination files.
2. Reads chunks from the source and writes them to the destination.
3. Closes both file descriptors.

Error Handling:

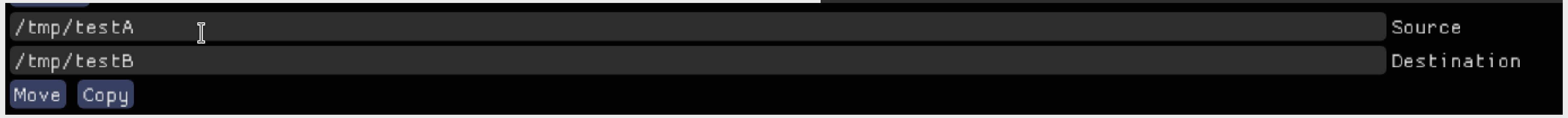
- Handles errors during opening, reading, or writing.



```cpp
int copy_file(const char *source, const char *destination) {
    int source_fd = open(source, O_RDONLY);
    if (source_fd == -1) {
        perror("open source file failed");
        return errno;
    }

    int dest_fd = open(destination, O_CREAT | O_WRONLY, 0666);
    if (dest_fd == -1) {
        perror("open destination file failed");
        close(source_fd);
        return errno;
    }

    char buffer[1024];
    ssize_t bytes_read;
    while ((bytes_read = read(source_fd, buffer, sizeof(buffer))) > 0) {
        if (write(dest_fd, buffer, bytes_read) == -1) {
            perror("write to destination file failed");
            close(source_fd);
            close(dest_fd);
            return errno;
        }
    }

    if (bytes_read == -1) {
        perror("read from source file failed");
    } else {
        std::cout << "File copied from " << source << " to " << destination << std::endl;
    }

    close(source_fd);
    close(dest_fd);
    return 0;
}
```



```
Status Message:
Copied successfully: /tmp/testA -> /tmp/testB
```



```
/tmp/testA    Source
/tmp/testB    Destination
```

Create File

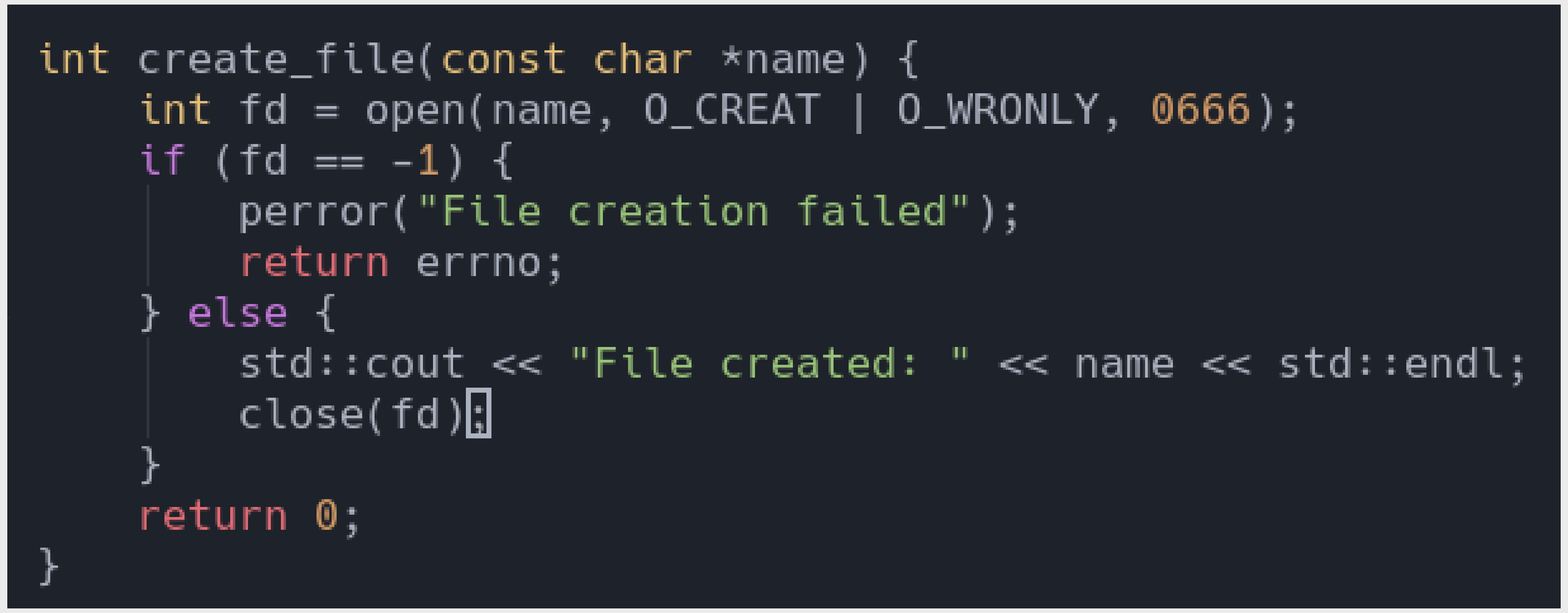
Creates a new file with write permissions (0666)

Logic:

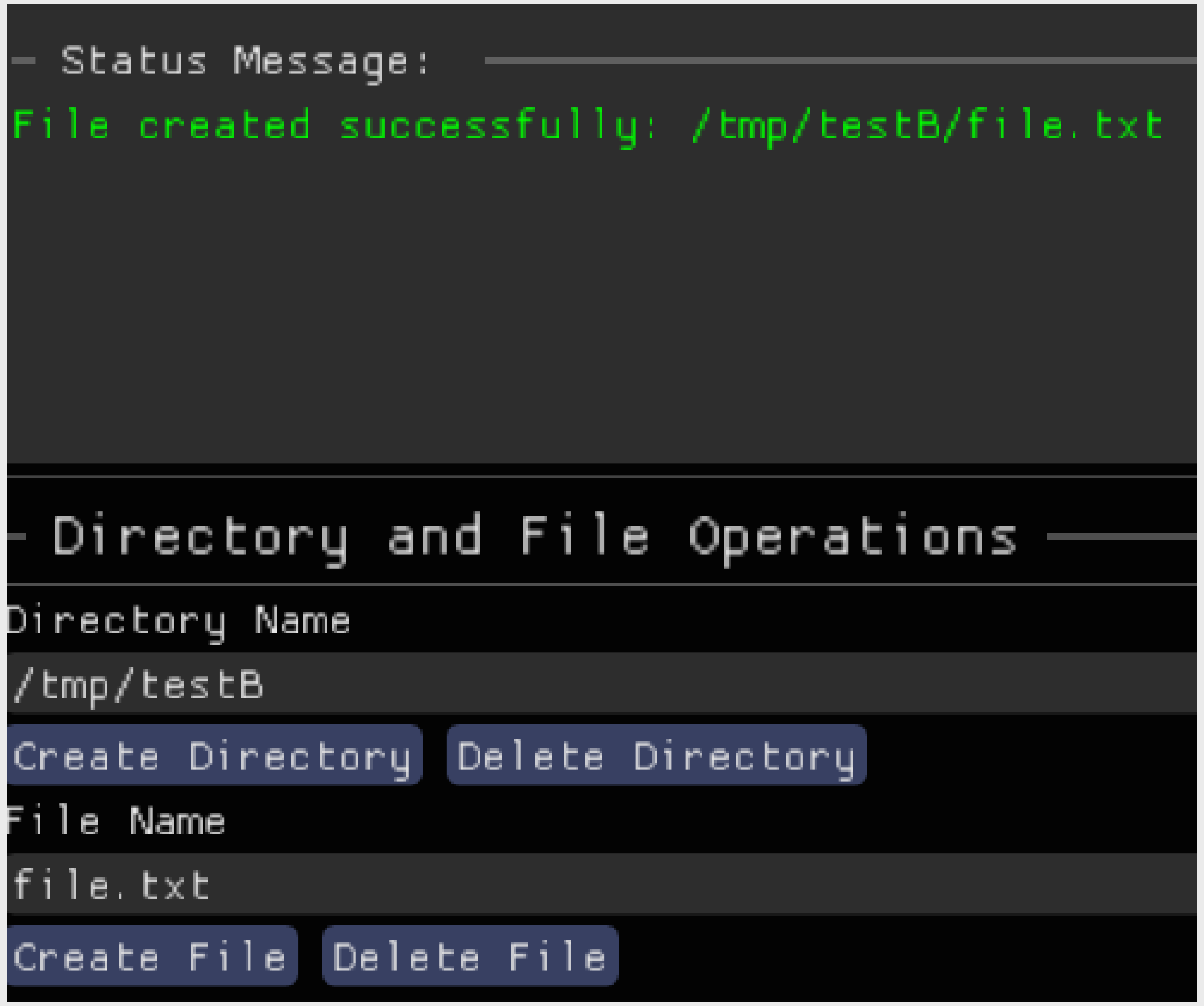
1. Opens the file with open using O_CREAT | O_WRONLY.
2. Prints a success message.
3. Closes the file descriptor after creation.

Error Handling:

- Uses errno and perror to handle errors like permission issues



```
int create_file(const char *name) {
    int fd = open(name, O_CREAT | O_WRONLY, 0666);
    if (fd == -1) {
        perror("File creation failed");
        return errno;
    } else {
        std::cout << "File created: " << name << std::endl;
        close(fd);
    }
    return 0;
}
```



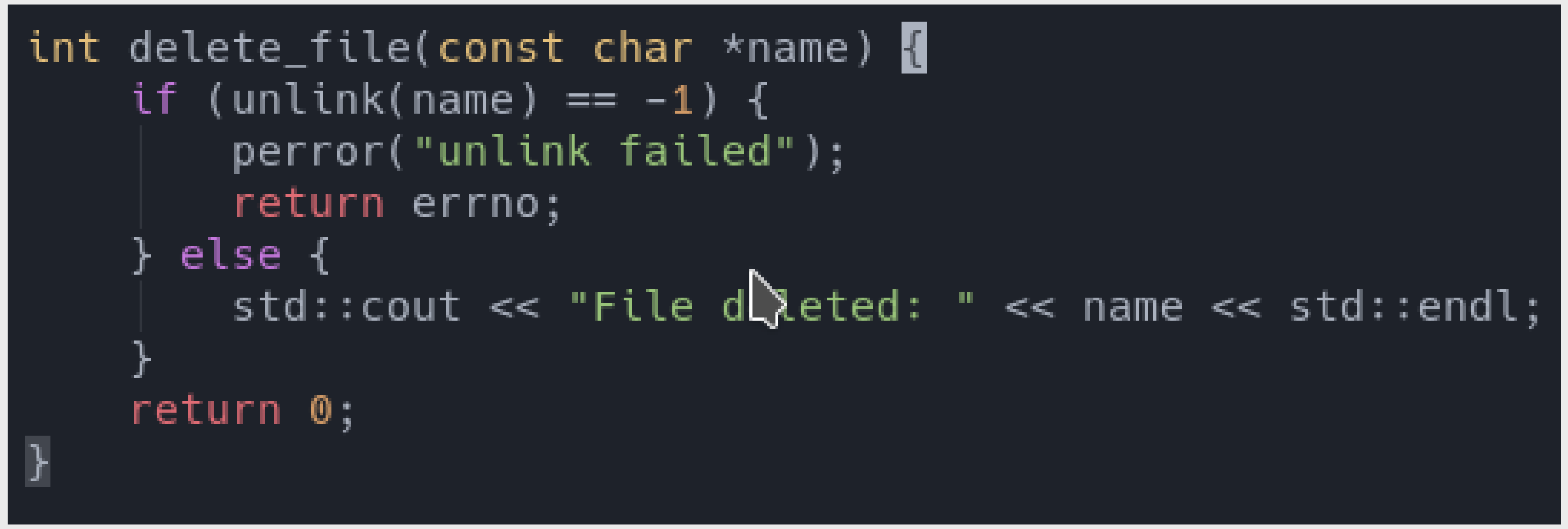
```
Status Message:
File created successfully: /tmp/testB/file.txt

Directory and File Operations
Directory Name
/tmp/testB
File Name
file.txt
```

Delete File

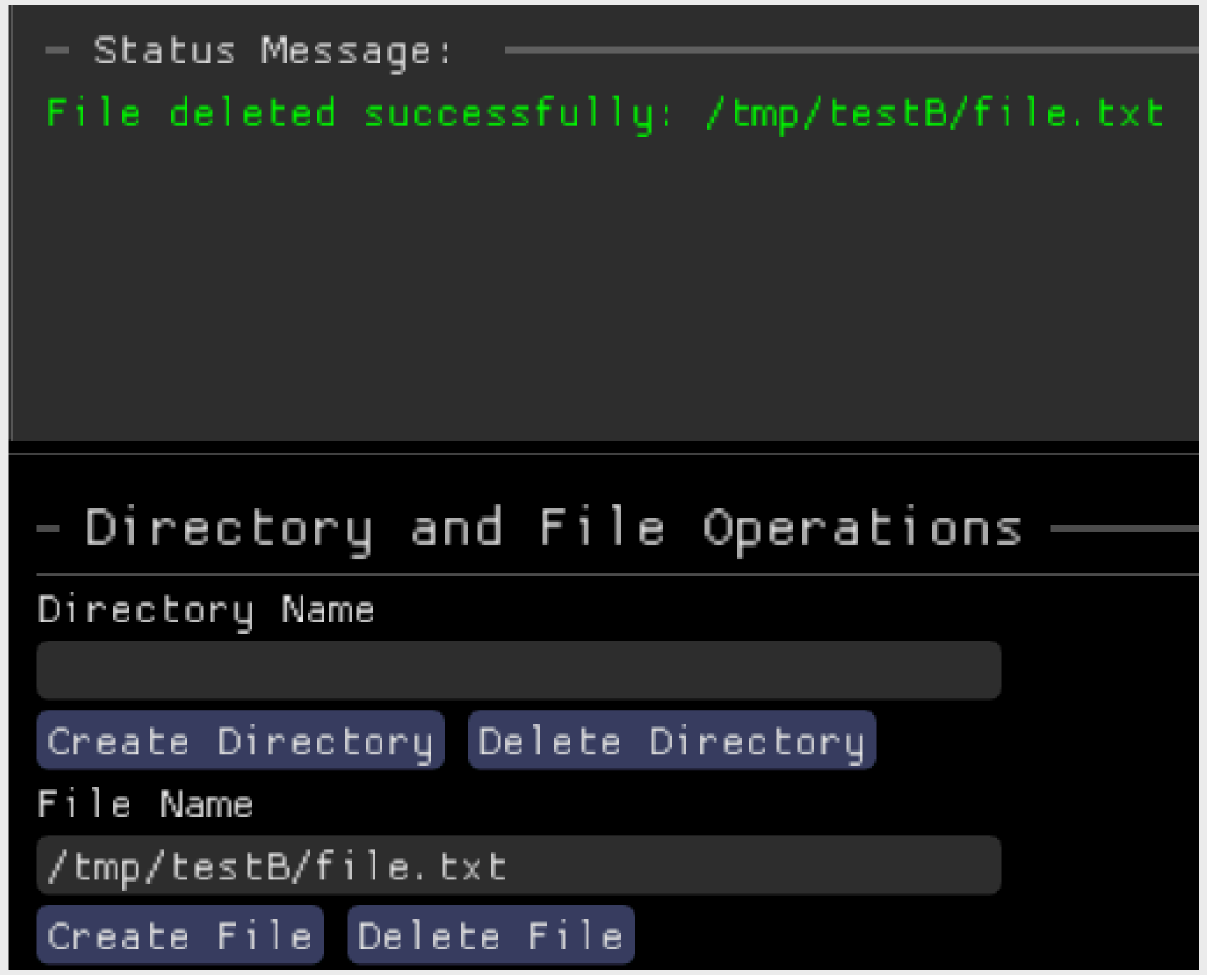
Deletes the specified file.

Logic:

1. Uses unlink to delete the file.
2. Prints success or error messages.



```
int delete_file(const char *name) {
    if (unlink(name) == -1) {
        perror("unlink failed");
        return errno;
    } else {
        std::cout << "File deleted: " << name << std::endl;
    }
    return 0;
}
```



```
Status Message:
File deleted successfully: /tmp/testB/file.txt
```

Directory and File Operations

Directory Name

File Name: /tmp/testB/file.txt

Writing to File

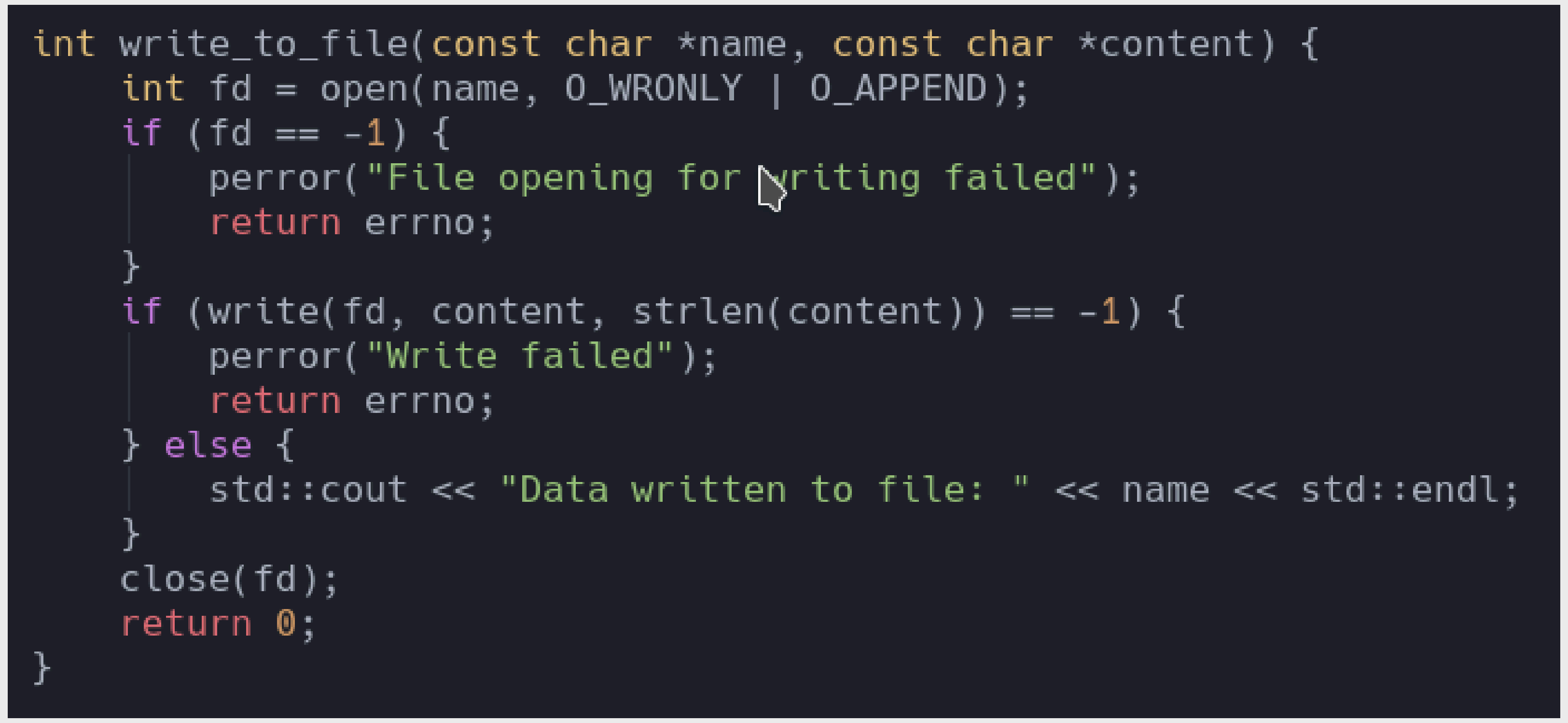
Appends content to the specified file.

Logic:

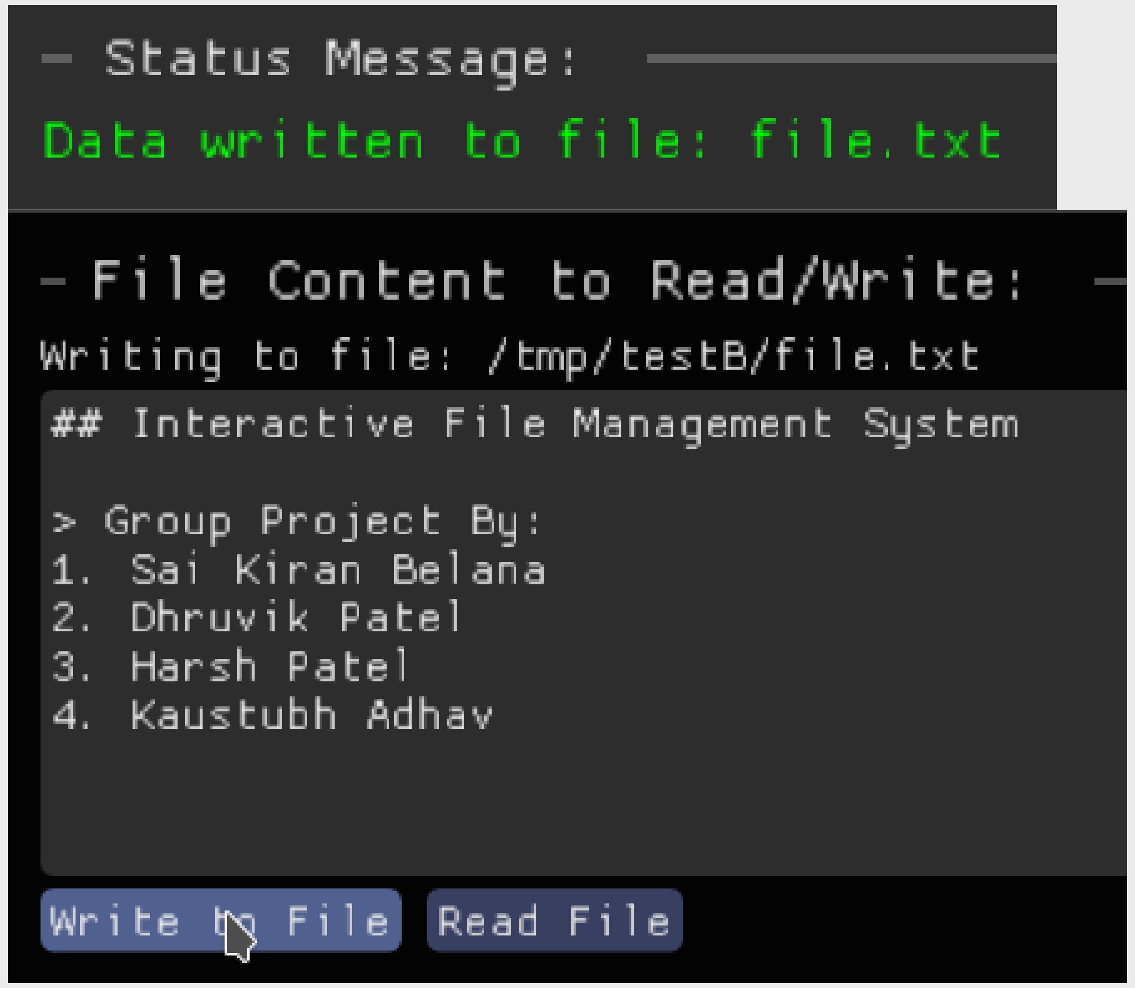
1. Opens the file in append mode (O_WRONLY | O_APPEND).
2. Writes the content using write.
3. Closes the file descriptor.

Error Handling:

- Checks errors during opening or writing.



```
int write_to_file(const char *name, const char *content) {
    int fd = open(name, O_WRONLY | O_APPEND);
    if (fd == -1) {
        perror("File opening for writing failed");
        return errno;
    }
    if (write(fd, content, strlen(content)) == -1) {
        perror("Write failed");
        return errno;
    } else {
        std::cout << "Data written to file: " << name << std::endl;
    }
    close(fd);
    return 0;
}
```



```
Status Message:
Data written to file: file.txt

File Content to Read/Write:
Writing to file: /tmp/testB/file.txt
## Interactive File Management System

> Group Project By:
1.  Sai Kiran Belana
2.  Dhruvik Patel
3.  Harsh Patel
4.  Kaustubh Adhav
```

Read from File

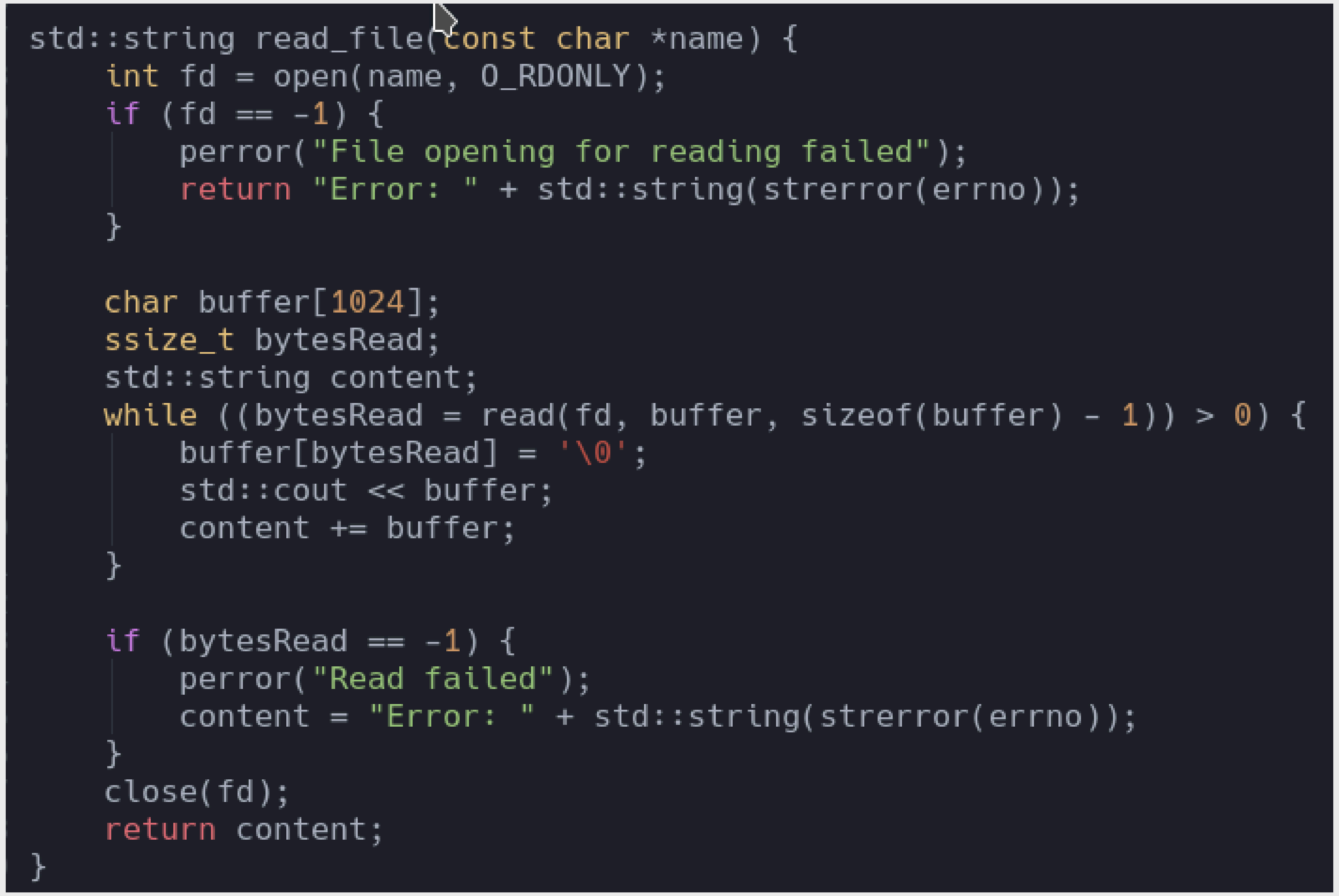
Reads the entire content of a file.

Logic:

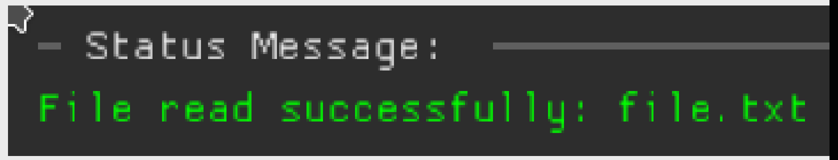
1. Opens the file in read mode (O_RDONLY).
2. Reads in chunks of 1 KB using read.
3. Accumulates the content in a string and returns it.
4. Closes the file descriptor.

Error Handling:

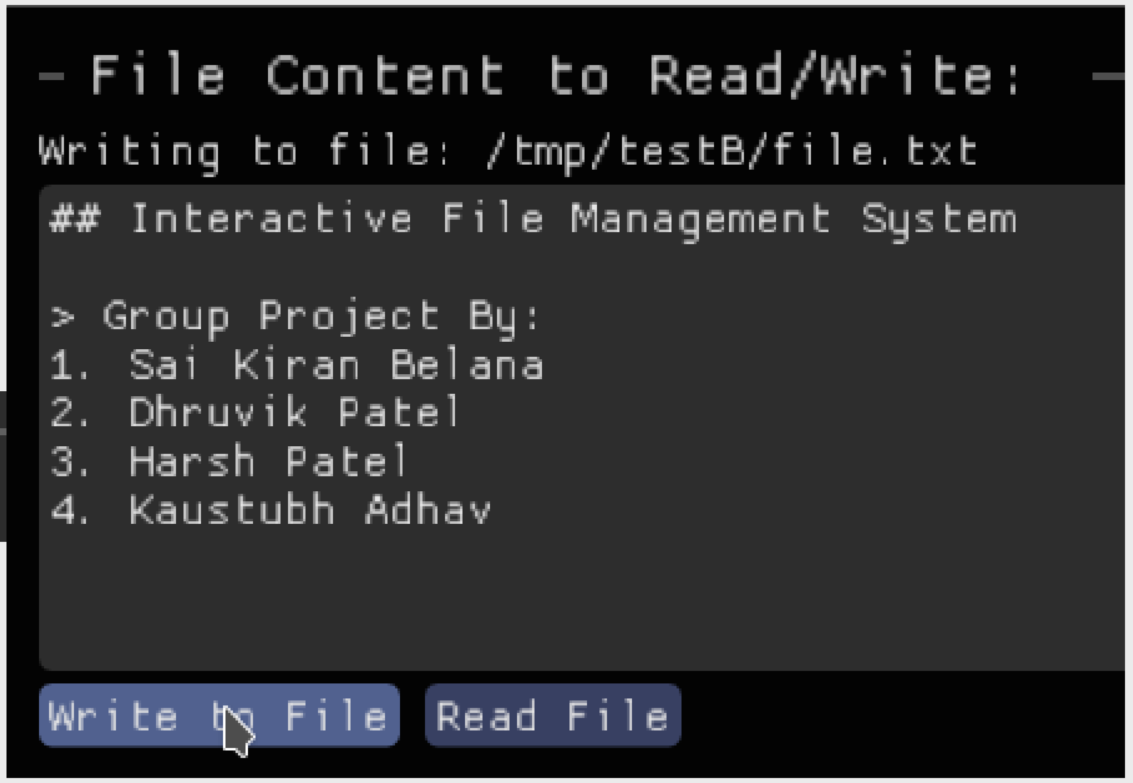
- Handles errors during opening and reading.



```
std::string read_file(const char *name) {
    int fd = open(name, O_RDONLY);
    if (fd == -1) {
        perror("File opening for reading failed");
        return "Error: " + std::string(strerror(errno));
    }

    char buffer[1024];
    ssize_t bytesRead;
    std::string content;
    while ((bytesRead = read(fd, buffer, sizeof(buffer) - 1)) > 0) {
        buffer[bytesRead] = '\0';
        std::cout << buffer;
        content += buffer;
    }

    if (bytesRead == -1) {
        perror("Read failed");
        content = "Error: " + std::string(strerror(errno));
    }
    close(fd);
    return content;
}
```



```
Status Message:
File read successfully: file.txt
```



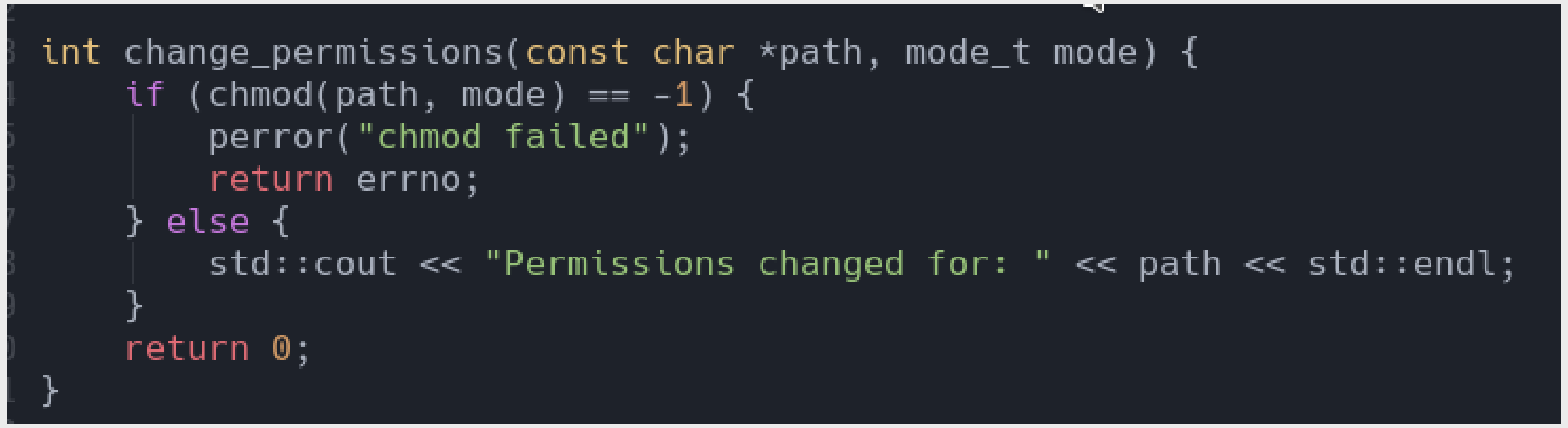
```
File Content to Read/Write:
Writing to file: /tmp/testB/file.txt
## Interactive File Management System

> Group Project By:
1.  Sai Kiran Belana
2.  Dhruvik Patel
3.  Harsh Patel
4.  Kaustubh Adhav
```

Change File/Dir Permissions

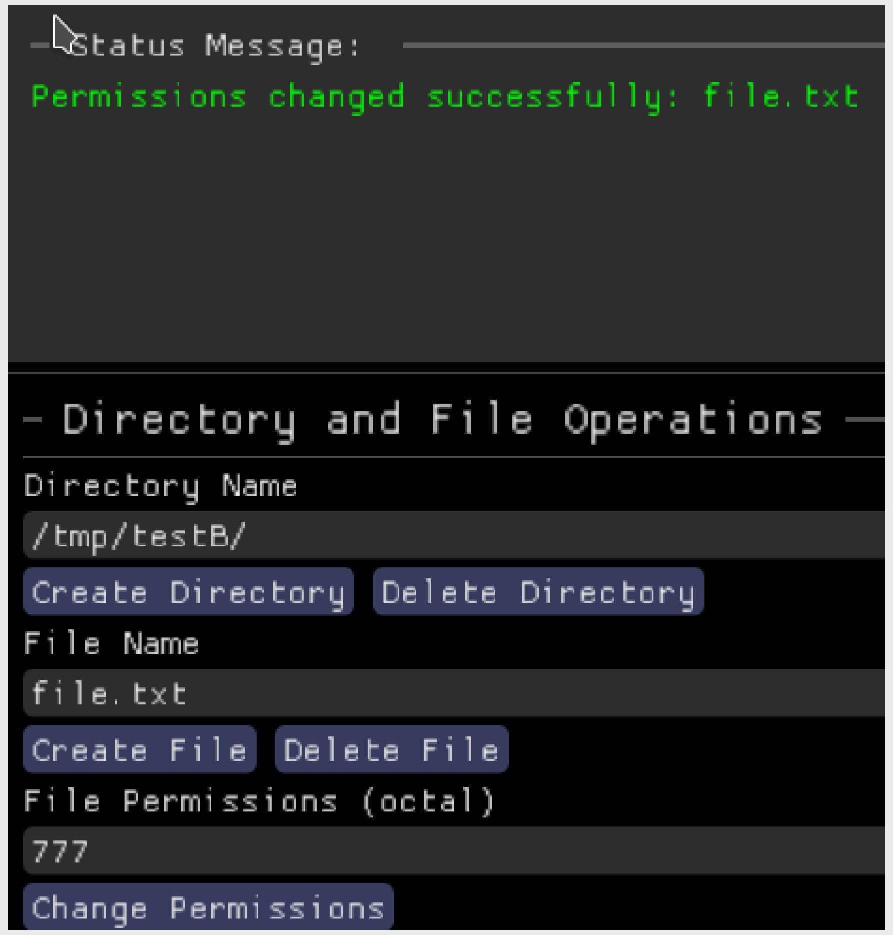
Updates file or directory permissions.

Logic:

- Uses chmod to set the 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;
}
```



```
Status Message:
Permissions changed successfully: file.txt
```

Directory and File Operations

Directory Name: /tmp/testB/

File Name: file.txt

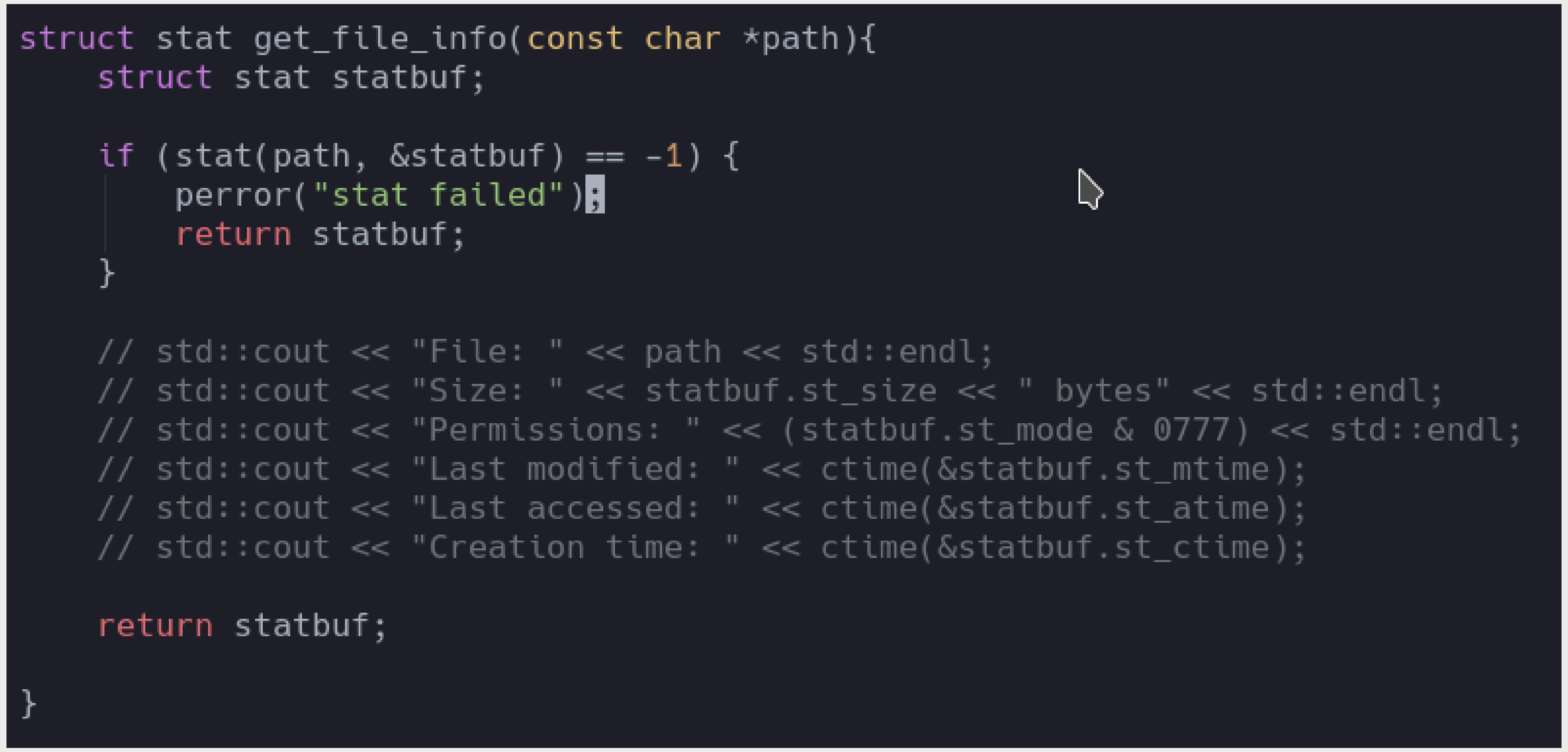
File Permissions (octal): 777

Get File Information

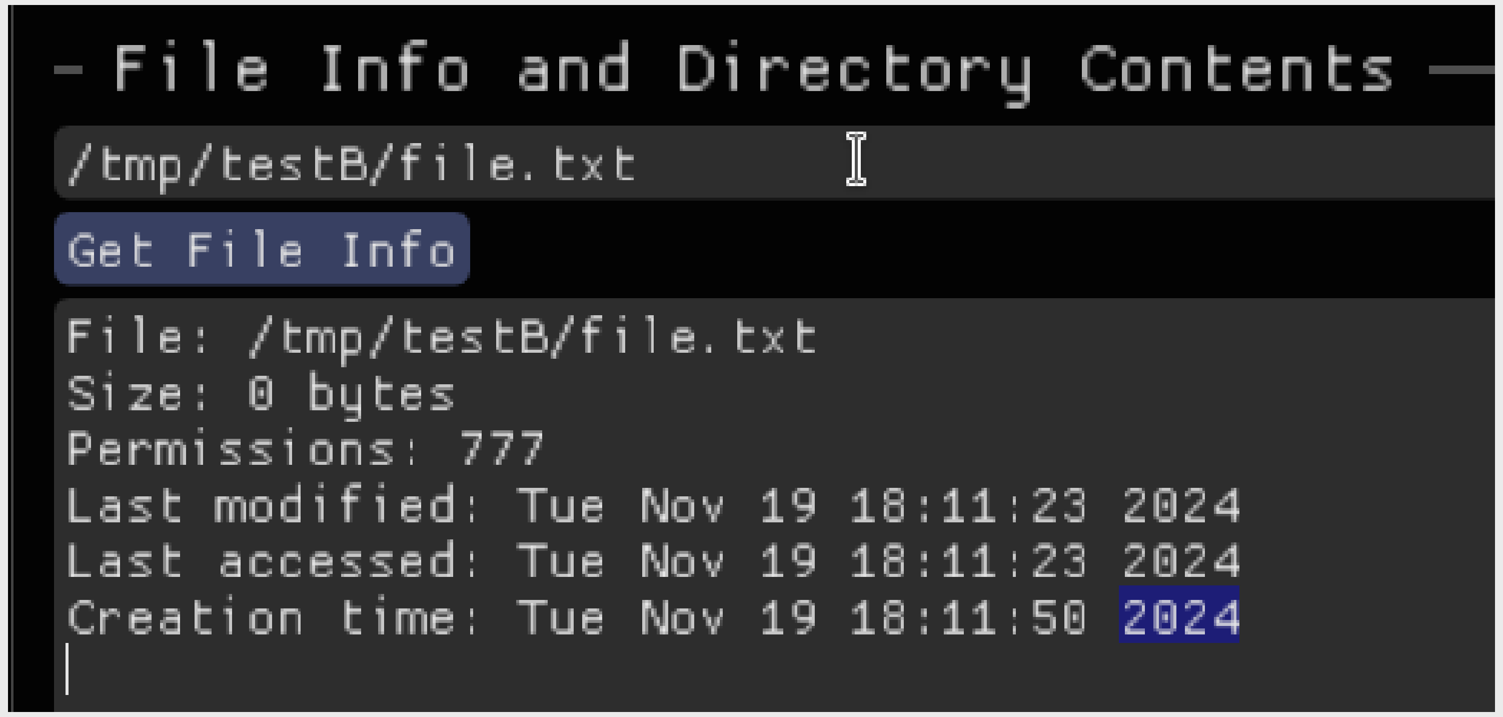
Retrieves metadata (size, permissions, timestamps) for a file or directory.

Logic:

1. Uses stat to fill a stat structure with metadata.
2. Returns the structure.



```
struct stat get_file_info(const char *path){
    struct stat statbuf;

    if (stat(path, &statbuf) == -1) {
        perror("stat failed");
        return statbuf;
    }

    // std::cout << "File: " << path << std::endl;
    // std::cout << "Size: " << statbuf.st_size << " bytes" << std::endl;
    // std::cout << "Permissions: " << (statbuf.st_mode & 0777) << std::endl;
    // std::cout << "Last modified: " << ctime(&statbuf.st_mtime);
    // std::cout << "Last accessed: " << ctime(&statbuf.st_atime);
    // std::cout << "Creation time: " << ctime(&statbuf.st_ctime);

    return statbuf;

}
```



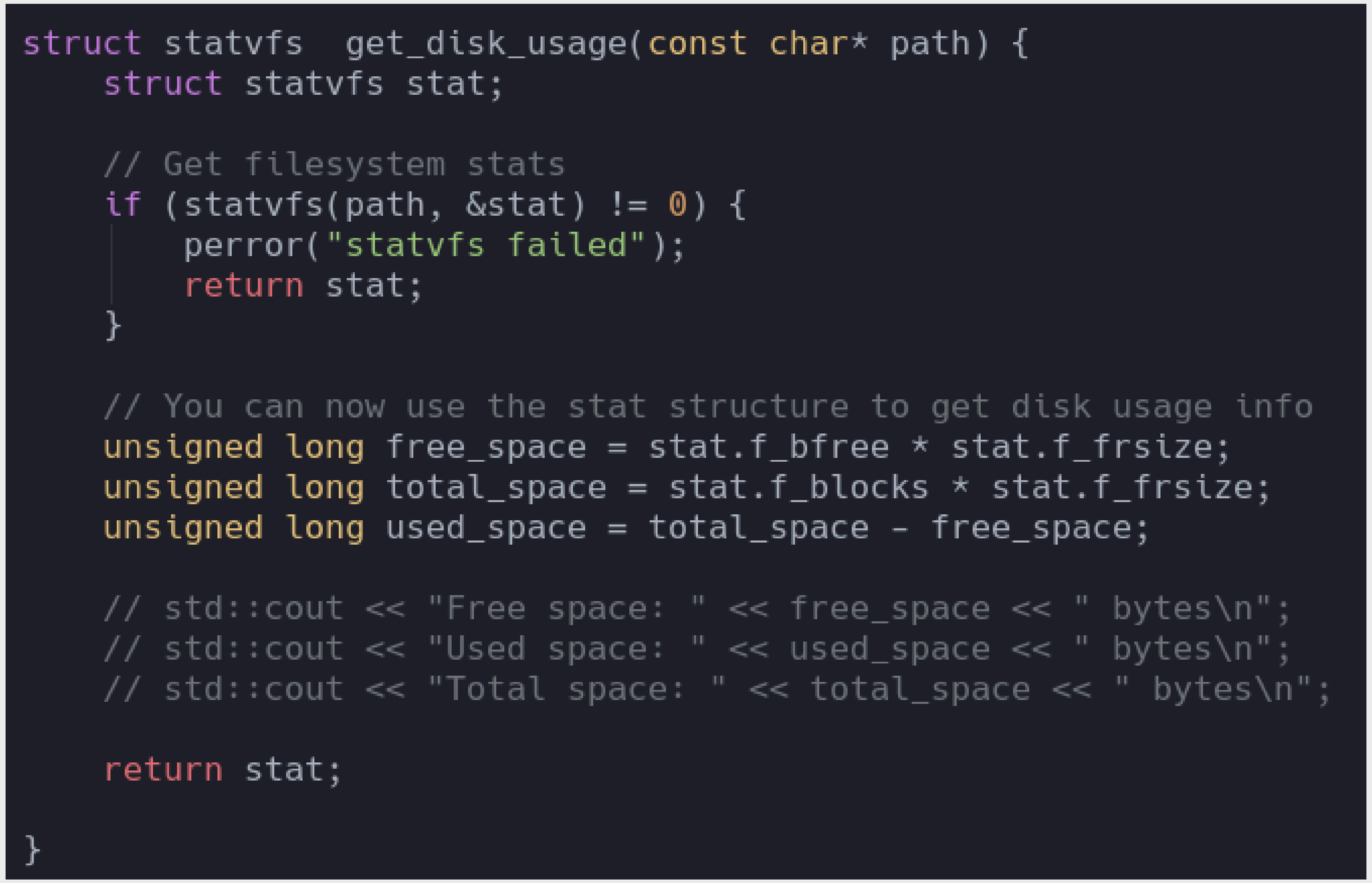
```
File: /tmp/testB/file.txt
Size: 0 bytes
Permissions: 777
Last modified: Tue Nov 19 18:11:23 2024
Last accessed: Tue Nov 19 18:11:23 2024
Creation time: Tue Nov 19 18:11:50 2024
```

Disk Usage:

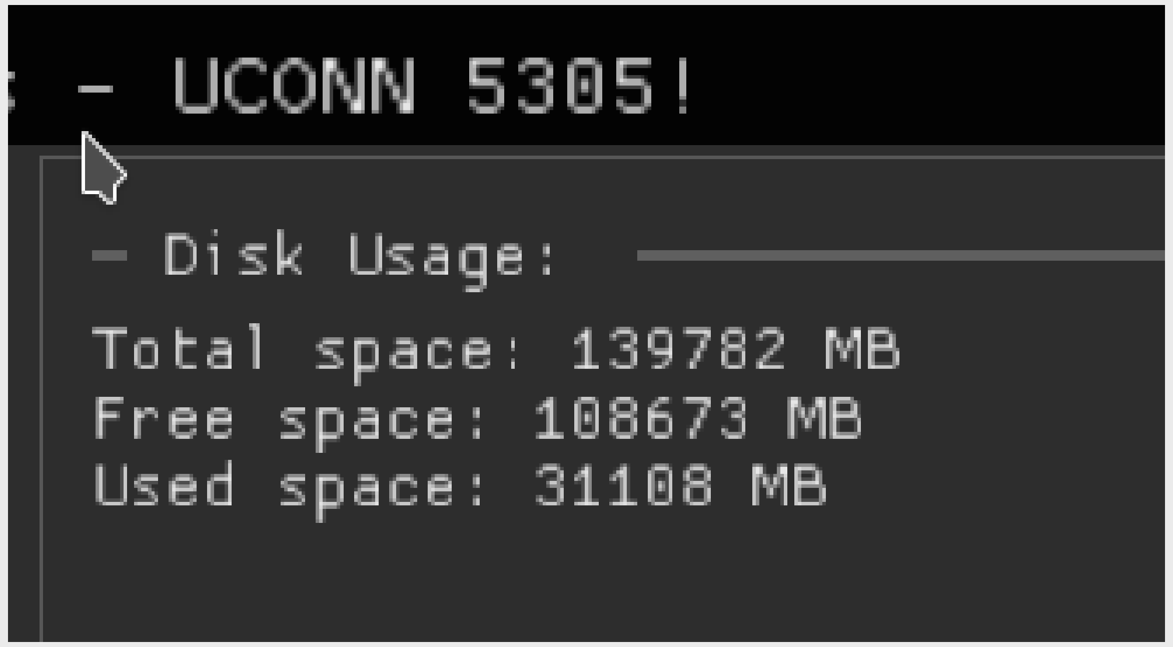
Provides disk usage statistics for the filesystem containing the path.

Logic:
Uses statvfs to retrieve filesystem stats and calculates free, used, and total space.

Output:
Shows the live Disk Usage Status – thanks to DearImGUI, since it renders in real-time,we can call in the function in real-time



```
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;

}
```



```
- UCONN 5305!

- Disk Usage:
Total space: 139782 MB
Free space: 108673 MB
Used space: 31108 MB
```

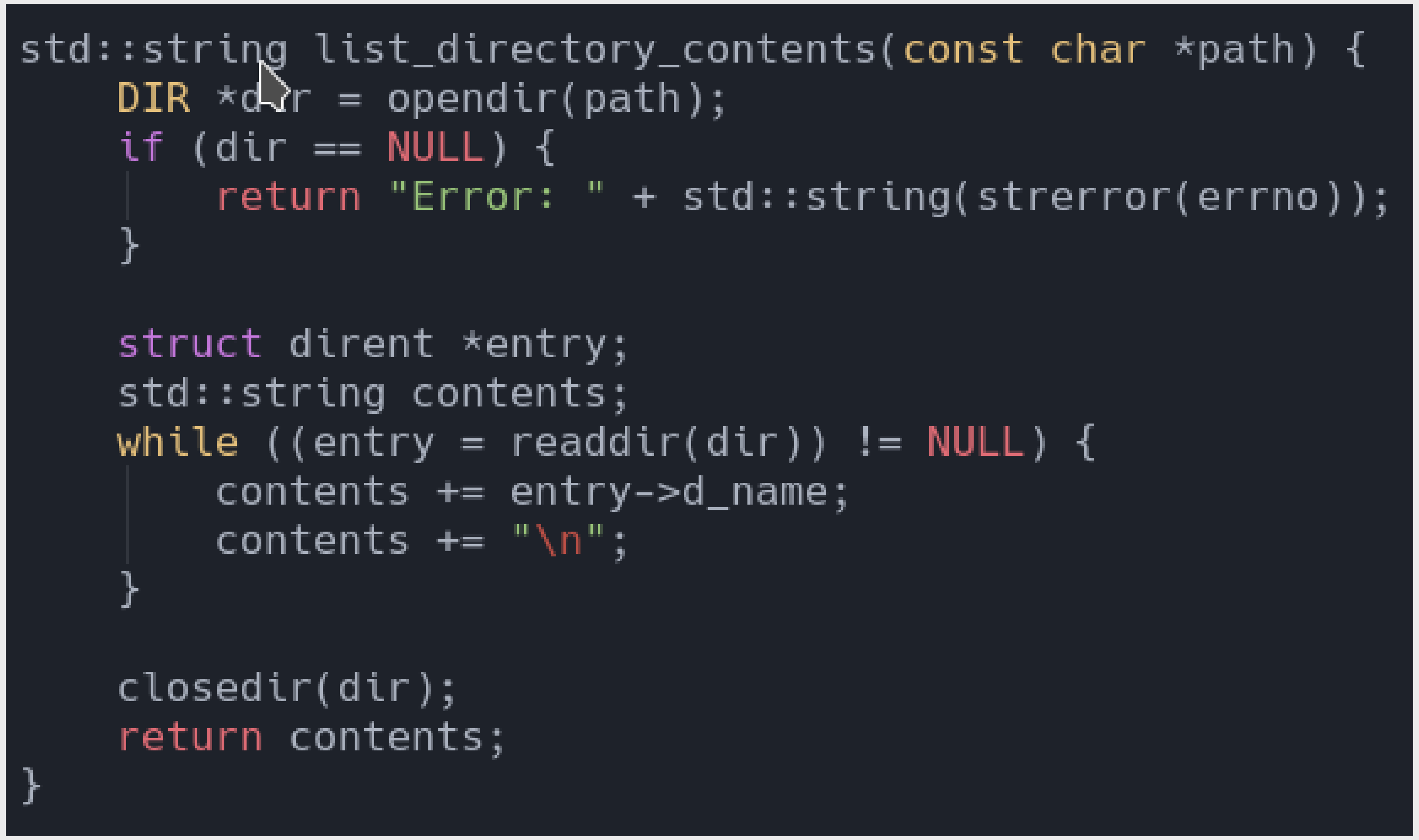
List Directory Contents

Lists all files and directories within a specified directory.

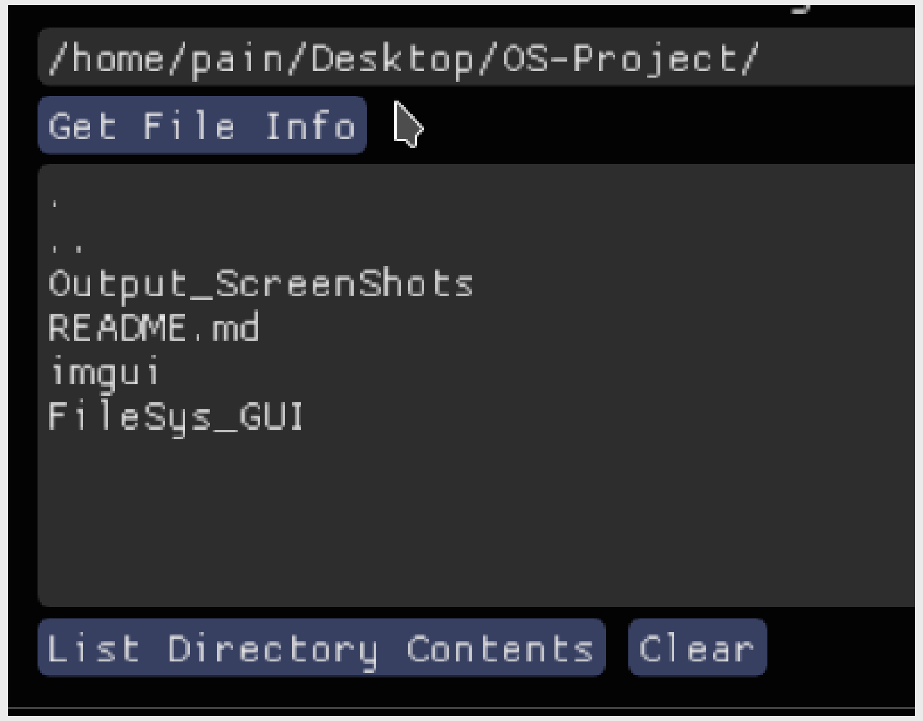
Logic:

1. Opens the directory with opendir.
2. Reads entries with readdir.
3. Appends each entry name to a string with newline separation.
4. Closes the directory before returning the result.

Return Value: A string containing all entry names, or an error message.



```
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;
}
```



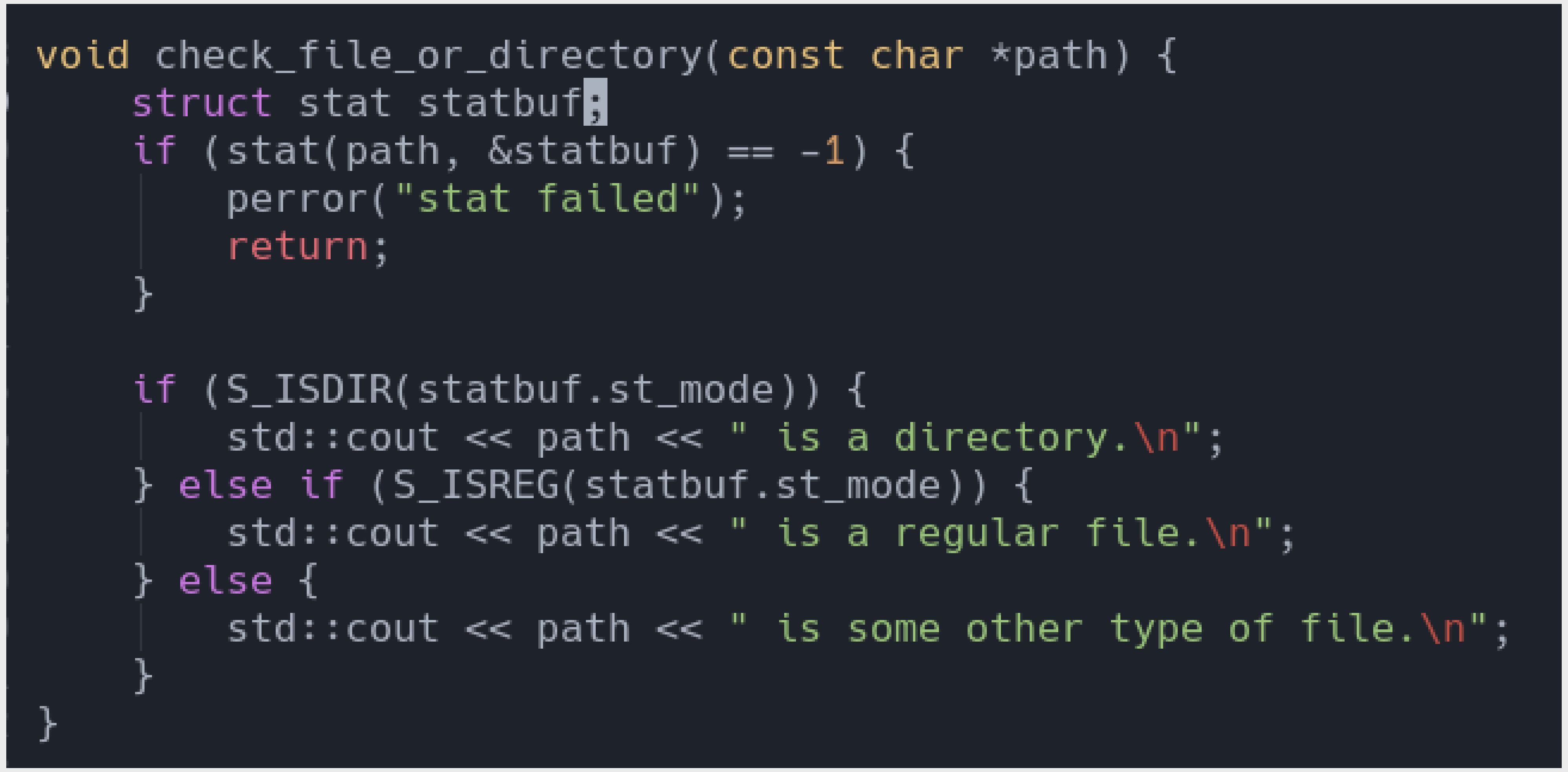
```
/home/pain/Desktop/OS-Project/
.
..
Output_ScreenShots
README.md
imgui
FileSys_GUI
```

Check if path is a file/Dir

Determines if the path is a file, directory, or another type.

Logic:

- Uses stat to retrieve the mode and checks with S_ISDIR or S_ISREG.



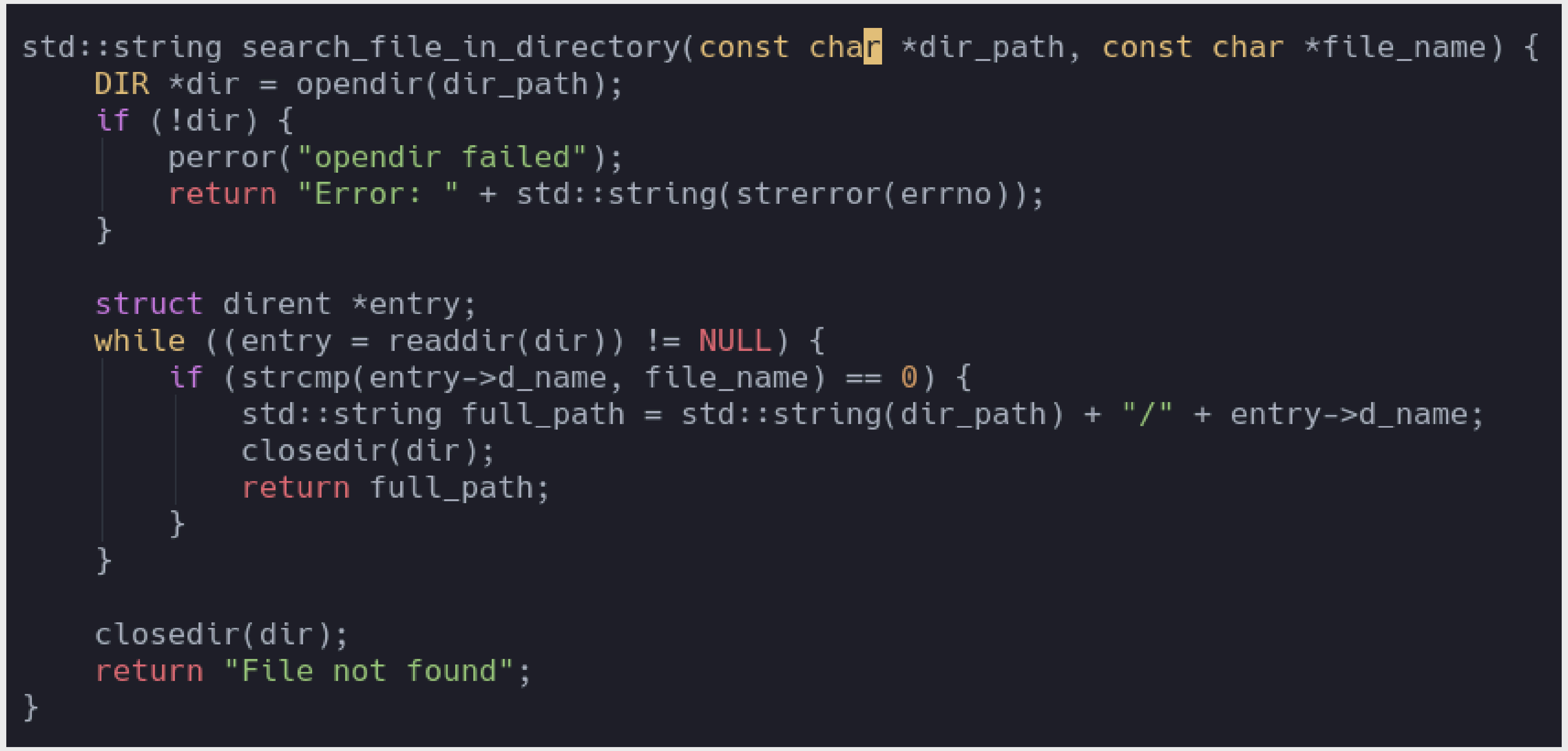
```
void check_file_or_directory(const char *path) {
    struct stat statbuf;
    if (stat(path, &statbuf) == -1) {
        perror("stat failed");
        return;
    }

    if (S_ISDIR(statbuf.st_mode)) {
        std::cout << path << " is a directory.\n";
    } else if (S_ISREG(statbuf.st_mode)) {
        std::cout << path << " is a regular file.\n";
    } else {
        std::cout << path << " is some other type of file.\n";
    }
}
```

Search for File/Dir

Searches for a file within a directory.

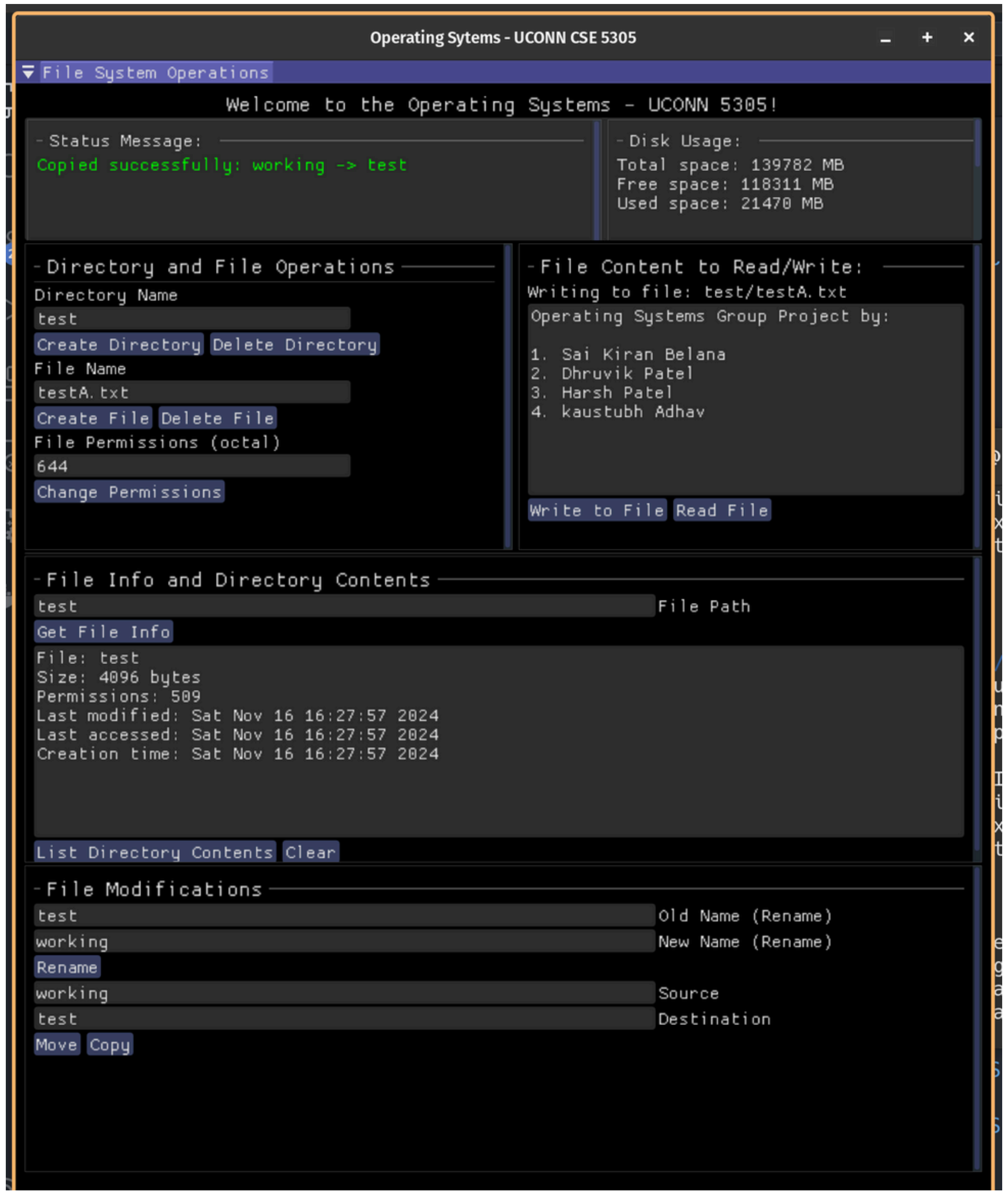
Logic:

- Iterates through directory entries and compares each name with file_name.



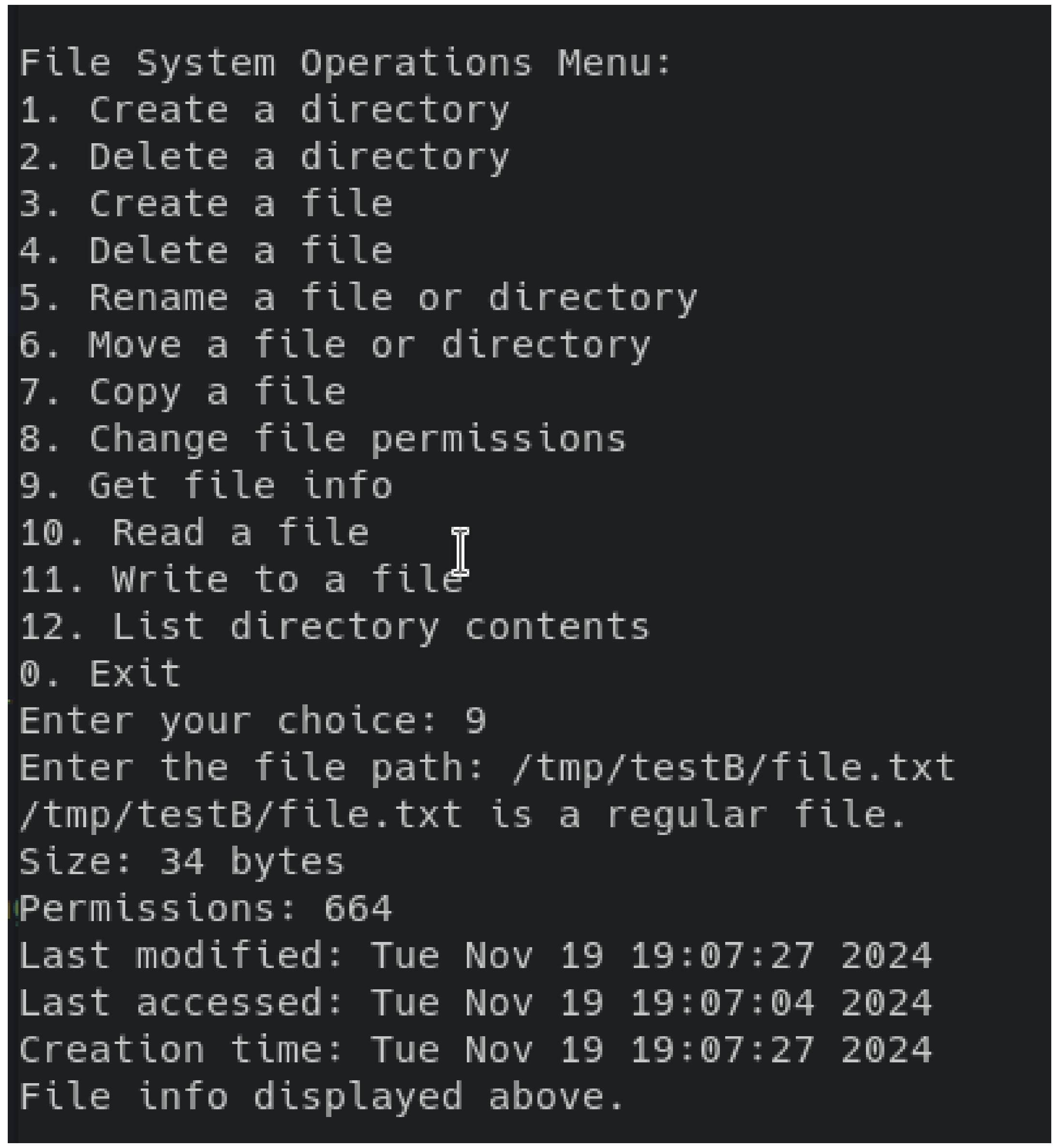
```
std::string search_file_in_directory(const char *dir_path, const char *file_name) {
    DIR *dir = opendir(dir_path);
    if (!dir) {
        perror("opendir failed");
        return "Error: " + std::string(strerror(errno));
    }

    struct dirent *entry;
    while ((entry = readdir(dir)) != NULL) {
        if (strcmp(entry->d_name, file_name) == 0) {
            std::string full_path = std::string(dir_path) + "/" + entry->d_name;
            closedir(dir);
            return full_path;
        }
    }

    closedir(dir);
    return "File not found";
}
```

Output:



```
Operating Sytems - UCONN CSE 5305
File System Operations
Welcome to the Operating Systems - UCONN 5305!

Status Message:
Copied successfully: working -> test

Disk Usage:
Total space: 139782 MB
Free space: 118311 MB
Used space: 21470 MB

Directory and File Operations
Directory Name
test
Create Directory  Delete Directory
File Name
testA.txt
Create File  Delete File
File Permissions (octal)
644
Change Permissions

File Content to Read/Write:
Writing to file: test/testA.txt
Operating Systems Group Project by:

1.  Sai Kiran Belana
2.  Dhruvik Patel
3.  Harsh Patel
4.  kaustubh Adhav
Write to File  Read File

File Info and Directory Contents
test                                  File Path
Get File Info
File: test
Size: 4096 bytes
Permissions: 509
Last modified: Sat Nov 16 16:27:57 2024
Last accessed: Sat Nov 16 16:27:57 2024
Creation time: Sat Nov 16 16:27:57 2024
List Directory Contents  Clear

File Modifications
test                                  Old Name (Rename)
working                               New Name (Rename)
Rename
working                               Source
test                                  Destination
Move  Copy
```

Testing case:
Terminal application -> tui.cpp



```
File System Operations Menu:
1. Create a directory
2. Delete a directory
3. Create a file
4. Delete a file
5. Rename a file or directory
6. Move a file or directory
7. Copy a file
8. Change file permissions
9. Get file info
10. Read a file
11. Write to a file
12. List directory contents
0. Exit
Enter your choice: 9
Enter the file path: /tmp/testB/file.txt
/tmp/testB/file.txt is a regular file.
Size: 34 bytes
Permissions: 664
Last modified: Tue Nov 19 19:07:27 2024
Last accessed: Tue Nov 19 19:07:04 2024
Creation time: Tue Nov 19 19:07:27 2024
File info displayed above.
```

Future Work

Add functions -> GUI

1. cd
2. search_file_in_directory
3. check_if_exists

Thank you