CS 3502: Operating Systems

Project 3

File System Implementation

C **R** **U** **D**

OwlTech File Systems Division

Department of Computer Science
College of Computing and Software Engineering
Kennesaw State University

KSU CCSE CS

Contents

1 Introduction

Welcome to OwlTech File Systems Division!

After your successful work on threading, IPC, and scheduling, the File Systems Division needs your expertise. Our enterprise systems require a robust file management solution with a modern graphical interface. You'll build a file manager that demonstrates your understanding of OS file system operations while providing a user-friendly experience.
This project bridges the gap between low-level system calls and high-level user interfaces, showing how operating systems abstract file operations for applications.

1.1 Learning Objectives

Upon completion of this project, you will be able to:

- Understand how operating systems manage files and directories
- Implement CRUD (Create, Read, Update, Delete) operations
- Work with file metadata and attributes
- Build a graphical user interface for system operations
- Handle errors and edge cases in file operations
- Document and present technical work professionally

1.2 Timeline

Milestone	Date	Description
Project Released	November 20	Specification available
Thanksgiving Break	Nov 25-29	University closed
Due Date	**December 4**	Submit by 11:59 PM
Late Deadline	December 8	Drop box closes (with penalty)

Table 1: Project Timeline

2 Project Overview

2.1 What You'll Build

You will create a file management system with a graphical user interface that demonstrates your understanding of operating system file operations. Your application will:

- Provide a visual interface for file system navigation
- Implement all basic file operations (Create, Read, Update, Delete)
- Support directory operations and navigation
- Display file metadata and properties

- Handle errors gracefully with user feedback
- Maintain data integrity during operations

2.2 Technology Choices

Flexibility is Key!

You may use any programming language and GUI framework you're comfortable with. Choose based on your experience and what will best demonstrate your skills. The important part is the functionality, not the specific technology.

Language	**GUI Framework Options**
C/C++	Qt, GTK+, or Windows Forms (via C++/CLI)
Python	Tkinter, PyQt5, Kivy, or wxPython
Java	Swing or JavaFX
C#	Windows Forms or WPF
Web-based	Electron with Node.js
Other	Any language with GUI support

Table 2: Technology Stack Options

3 Functional Requirements

3.1 Core Operations (Required)

Your file manager must implement all of the following operations:

Operation	**Description**	**User Interface**
CREATE	Create new files and directories	Dialog or form for name/content
READ	View file contents	Display area or editor window
UPDATE	Modify existing files	Edit capability with save function
DELETE	Remove files and directories	Confirmation dialog, then removal
RENAME	Change file/directory names	Input dialog for new name
NAVIGATE	Browse directory structure	Tree view or list with folders

Table 3: Required Core Operations

3.2 Additional Features (Optional but Encouraged)

Consider implementing these features for extra credit or to enhance your project:

- File search functionality
- Copy/paste operations
- File properties display (size, permissions, timestamps)
- Multiple file selection

- Drag and drop support
- Context menus (right-click)
- Keyboard shortcuts
- File type icons
- Recent files list
- Undo/redo operations

4 Implementation Guidelines

4.1 Minimum GUI Components

Your interface should include at least:

1. **File/Directory Display:** A list or tree view showing files and folders
2. **Action Buttons/Menu:** Clear ways to trigger CRUD operations
3. **Current Path Display:** Show where the user is in the file system
4. **Status/Feedback Area:** Inform users of operation results
5. **File Content Area:** For viewing/editing file contents (can be a separate window)

4.2 Error Handling

Robust Error Handling Required

Your application should gracefully handle all error conditions and display user-friendly messages, not raw system errors.

Common error scenarios to handle:

- File/directory already exists
- File not found
- Permission denied
- Disk space issues
- Invalid file names
- Files in use by other processes
- Network drive disconnections (if applicable)

4.3 Code Structure

Organize your code professionally:

- **Separation of Concerns:** Keep GUI code separate from file operations
- **Error Handling:** Consistent error checking and reporting
- **Comments:** Explain complex operations and design decisions
- **Constants:** Define magic numbers and strings as constants
- **Functions:** Break operations into logical, reusable functions

5 Testing Requirements

5.1 Test Scenarios

Test your application with these scenarios:

5.1.1 Basic Operations

1. Create a file with content
2. Read the file back
3. Update the content
4. Rename the file
5. Delete the file

5.1.2 Directory Operations

1. Create nested directories
2. Navigate up and down the tree
3. Delete empty and non-empty directories

5.1.3 Edge Cases

1. Very long file names
2. Special characters in names
3. Empty files
4. Large files (test with at least 1MB)
5. Attempting to delete system files (should fail gracefully)

6 Deliverables

Component	Requirements	Points
Source Code	• All source files • Build instructions • Dependencies list	100
Report (PDF)	• Implementation overview • Design decisions • Challenges faced • Testing results • 3-5 pages	40
Demo Video	• 3-5 minutes • Show all CRUD operations • Demonstrate error handling • Explain key features	40
Total		**180**

Table 4: Deliverables and Points Distribution

7 Report Guidelines

Your report should be a professional technical document (3-5 pages) containing:

7.1 Required Sections

1. **Introduction**
    - Brief project overview
    - Technologies chosen and why
2. **Design and Architecture**
    - High-level architecture diagram
    - Key design decisions
    - Data structures used
3. **Implementation**
    - Core algorithms/approaches
    - How you handle file operations
    - GUI framework integration
    - Error handling strategy

4. **Testing**
   - Test cases performed
   - Results and any issues found
   - Performance observations
5. **Challenges and Solutions**
   - Technical difficulties encountered
   - How you solved them
   - What you learned
6. **Conclusion**
   - Project summary
   - Future improvements
   - Reflection on learning outcomes

8 Demo Video Guidelines

Create a 3-5 minute video demonstrating your file manager:

8.1 Video Structure

1. **Introduction (30 seconds)**
   - Your name
   - Brief overview of your implementation
   - Technologies used
2. **Demonstration (2-3 minutes)**
   - Create a new file with content
   - View the file
   - Edit and save changes
   - Rename the file
   - Create a directory
   - Move/copy file to new directory
   - Delete operations
   - Show error handling (try invalid operations)
3. **Code Walkthrough (1 minute)**
   - Show key functions briefly
   - Highlight interesting implementation details
4. **Conclusion (30 seconds)**
   - Summarize what was shown
   - Mention any special features

Video Recording Tips

- Use screen recording software (OBS Studio is free)
- Ensure audio is clear
- Keep the pace steady - don't rush
- Have a script or outline ready

9 Grading Rubric

Category	Criteria	Points
Core Functionality	• Create operations work correctly (20) • Read operations work correctly (20) • Update operations work correctly (20) • Delete operations work correctly (20) • Directory navigation works (20)	100
User Interface	• Clean, intuitive design (10) • Proper feedback to user (10) • Error messages displayed appropriately (10)	30
Code Quality	• Well-organized code structure (10) • Comments and documentation (5) • Error handling (5)	20
Report	• Complete and professional (10) • Technical depth (5) • Reflection and analysis (5)	20
Demo Video	• Shows all required operations (5) • Clear presentation (3) • Within time limit (2)	10
Total		**180**

Table 5: Detailed Grading Rubric

10 Submission Instructions

Submit to D2L by the due date (see D2L)

Late submissions will incur a 10% penalty per day. The drop box closes on December 8, and no submissions will be accepted after that date.

10.1 Required Submission Components

1. **Source Code:**
   - ZIP file containing all source files
   - Include a README with build/run instructions
   - List all dependencies
2. **Report:**
   - PDF format (CS3502_P3_YourName.pdf)
   - 3-5 pages
   - Professional formatting
3. **Demo Video:**
   - Upload to YouTube (unlisted) or similar
   - Provide link in D2L submission comments
   - Ensure link works before submitting

11 Tips for Success

11.1 Development Approach

Start Simple!

Get basic file listing working first, then add operations one at a time. Test each operation thoroughly before moving on.

1. Create basic GUI layout
2. Implement file listing
3. Add create file functionality
4. Implement read/display
5. Add update capabilities
6. Implement delete with confirmation
7. Add directory operations
8. Polish UI and error handling

11.2 Common Pitfalls to Avoid

- Not handling permissions properly
- Forgetting to close file handles
- No confirmation for destructive operations
- Poor error messages (”Error occurred”)
- Hardcoded paths that won’t work on other systems
- Not testing with various file types and sizes

11.3 Resources

- **Python:** os, pathlib, shutil modules
- **C/C++:** filesystem library (C++17), dirent.h, sys/stat.h
- **Java:** java.nio.file package
- Your OS assignment code for reference
- Stack Overflow for specific issues

12 Getting Started - Example Structure

12.1 Basic File Operations (Pseudocode)

```
class FileManager:
    def create_file(self, path, content):
        if file_exists(path):
            show_error("File already exists")
            return False
        try:
            write_file(path, content)
            update_display()
            return True
        except Exception as e:
            show_error(str(e))
            return False

    def read_file(self, path):
        if not file_exists(path):
            show_error("File not found")
            return None
        try:
            content = read_file_content(path)
            return content
        except Exception as e:
            show_error(str(e))
            return None

    def update_file(self, path, new_content):
        # Similar pattern...
        pass

    def delete_file(self, path):
        if confirm_dialog("Delete " + path + "?"):
            # Perform deletion
            pass
```

Listing 1: Example File Manager Structure

Remember

Use your operating system's actual file APIs! The above is just pseudocode to illustrate the structure.

Good Luck!

Good Luck with Your Final Project!

Remember: Start early, test thoroughly, and ask questions if you need help.

The OwlTech File Systems Division looks forward to your implementation!

A Quick Reference - File Operations by Language

A.1 Python

```python
import os
import shutil
from pathlib import Path

# Create file
with open('file.txt', 'w') as f:
    f.write('content')

# Read file
with open('file.txt', 'r') as f:
    content = f.read()

# Update file
with open('file.txt', 'w') as f:
    f.write('new content')

# Delete file
os.remove('file.txt')

# Rename file
os.rename('old.txt', 'new.txt')

# Create directory
os.mkdir('new_dir')

# List directory
files = os.listdir('.')
```

A.2 C++

```cpp
#include <filesystem>
#include <fstream>

namespace fs = std::filesystem;

// Create file
std::ofstream file("file.txt");
file << "content";
file.close();

// Read file
std::ifstream infile("file.txt");
std::string content((std::istreambuf_iterator<char>(infile)),
                     std::istreambuf_iterator<char>());

// Delete file
fs::remove("file.txt");

// Rename file
fs::rename("old.txt", "new.txt");

// Create directory
```

```cpp
fs::create_directory("new_dir");

// List directory
for (const auto& entry : fs::directory_iterator(".")) {
    std::cout << entry.path() << std::endl;
}
```

A.3 Java

```java
import java.nio.file.*;
import java.io.IOException;

// Create file
Path path = Paths.get("file.txt");
Files.write(path, "content".getBytes());

// Read file
String content = new String(Files.readAllBytes(path));

// Update file
Files.write(path, "new content".getBytes());

// Delete file
Files.delete(path);

// Rename file
Files.move(Paths.get("old.txt"), Paths.get("new.txt"));

// Create directory
Files.createDirectory(Paths.get("new_dir"));

// List directory
Files.list(Paths.get(".")).forEach(System.out::println);
```