In-Memory File System Implementation

Fall 2024

Linux File Navigation Primer

Basic Concepts

In Unix-like systems, files are organized in a hierarchical directory structure. This structure starts at the root directory, represented by a forward slash (/), which contains all other files and directories.

Key Directory Concepts

- **Root Directory (/):** The top-level directory
- **Home Directory (˜):** Each user's personal directory
- **Current Directory (.):** The directory you're currently in
- **Parent Directory (..):** The directory one level up

Basic Commands

Common Linux Commands

```
$ pwd           # Print Working Directory
/home/user1

$ ls            # List Directory Contents
documents/  pictures/  file.txt

$ cd documents  # Change Directory
$ cd ..        # Go to parent directory
$ cd /         # Go to root directory

$ mkdir folder # Create Directory
$ touch file   # Create Empty File
$ rm file      # Remove File
```

1 Assignment Overview

This assignment requires implementing an in-memory file system using tree data structures in C++.

2 Core classes

```cpp
class FileSystemNode {
public:
    string name;
    bool isDirectory;
    vector<FileSystemNode*> children;
    FileSystemNode* parent;

    FileSystemNode(string name, bool isDir);
    ~FileSystemNode();
};
```

Listing 1: FileSystemNode Class

```cpp
class FileSystem {
private:
    FileSystemNode* root;
    FileSystemNode* currentDirectory;

public:
    FileSystem();
    ~FileSystem();

    void mkdir(const string& name);
    void cd(const string& path);
    void ls();
    void pwd();
    void touch(const string& name);
    void rm(const string& name);
};
```

Listing 2: FileSystem Class

3 Grading Structure

1. **Directory Operations (30 points)**
   - mkdir (10 points)
   - cd (20 points)
2. **File Operations (20 points)**
   - touch (10 points)
   - ls (10 points)
3. **Path and Removal Operations (40 points)**
   - pwd (15 points)
   - rm (25 points)

   Proper documentation - 10 points

4 Implementation Requirements

5 Required Operations and Implementation Hints

Quick Reference Examples

```
FileSystem fs;
fs.mkdir("docs");          // Create directory
fs.touch("file.txt");      // Create file
fs.cd("docs");            // Change directory
string listing = fs.ls(); // List contents
string path = fs.pwd();   // Show current path
fs.rm("file.txt");        // Remove file/directory
```

5.1 Core Operations

5.1.1 mkdir(const string& name)

Purpose: Create new directory
Key Points:

- Check for existing directory- if it already exists, return std::runtime_error("File already exists");
- Create node (isDirectory = true)
- Update parent/child links

Example:

```
fs.mkdir("docs");     // Success
fs.mkdir("docs");     // Error: Already exists
```

5.1.2 cd(const string& path)

Purpose: Navigate directories
Key Points:

- Handle "/", ".." cases
- Verify directory exists
- Update current directory
- if directory not found: throw std::runtime_error("Directory not found");

Example:

```
fs.cd("/");           // Root
fs.cd("..");          // Parent
fs.cd("docs");        // Child directory
```

5.1.3 ls()

Purpose: List directory contents
Key Points:

- Use stringstream
- Add "/" for directories
- Return formatted string

Example Output:

```
docs/
file.txt
images/
```

5.1.4 pwd()

Purpose: Show current path
Key Points:

- Build path from current to root
- Handle root directory case
- Format with leading/trailing "/"

Example:

```
/home/user/    // Multiple levels
/              // Root directory
```

5.1.5 touch(const string& name)

Purpose: Create new file
Key Points:

- Check for existing file. If a file with the same name exists:throw std::runtime_error("File already exists")
- Create node (isDirectory = false)
- Update parent/child links

Example:

```
fs.touch("note.txt");  // Success
fs.touch("note.txt");  // Error: Already exists
```

5.1.6 rm(const string& name)

Purpose: Remove file/directory
Key Points:

- Find target in current directory
- Delete node and all children
- Update parent's children vector
- if not found: throw std::runtime_error("File or directory not found")

Example:

```
fs.rm("file.txt");     // Remove file
fs.rm("docs");         // Remove directory and contents
```

5.2 Implementation Tips

Key Considerations

- Always maintain parent-child relationships
- Clean up memory in destructors
- Use consistent error handling
- Check edge cases (root, empty paths)
- Consider helper functions for common tasks

6 Testing Framework

6.1 Test Categories

Operation	**Points**
mkdir functionality	10
touch functionality	10
cd functionality	20
ls functionality	10
pwd functionality	15
rm functionality	25
Total	**90**

7 Submission Guidelines

1. Submit following files:
   - FileSystem.hpp
   - FileSystem.cpp
2. Code must compile without modifications
3. Include a makefile.
4. All files must be in a .zip named as {first_name}_{last_name}_p2.zip

8 Academic Integrity

All submitted work must be your own. Plagiarism will result in zero credit for the assignment.

9 Building and Testing

Compilation Instructions

```
# Compile the project
g++ -std=c++11 FileSystem.cpp FileSystemTester.cpp -o filesystem

# Run tests
./filesystem
```