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
$ cp source dest # Copy File or Directory
```

1 Assignment Overview

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

2 Required Operations and Implementation Hints

2.1 Core Operations

2.1.1 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
```

2.1.2 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
```

2.1.3 cp(const string& source, const string& destination)

Purpose: Copy files or directories
Key Points:

- Support copying both files and directories
- When When copying directories, recursively copy all contents
- Handle both absolute and relative paths for source and destination
- Check if source exists; if not: throw std::runtime_error("Source not found")
- Check if destination already exists; if yes: throw std::runtime_error("Destination already exists")
- Preserve directory structure when copying directories
- Maintain parent-child relationships in copied structures

Example:

```
fs.cp("file.txt", "backup.txt");  // Copy a file
fs.cp("docs", "docs_backup");     // Copy a directory and all contents
fs.cp("/home/user/file.txt", "/backup/file.txt");
// Using absolute paths
```

2.1.4 ls()

Purpose: List directory contents
Key Points:

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

Example Output:

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

2.1.5 pwd()

Purpose: Show current path
Key Points:

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

Example:

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

2.1.6 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
```

2.1.7 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
```

2.2 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

3 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}_S25_p2.zip

4 Academic Integrity

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

5 Building and Testing

Compilation Instructions

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

# Run tests
./filesystem
```