In-Memory File System Documentation

Introduction

Hello there! Welcome to the In-Memory File System (IMFS). This documentation is your guide to understanding the inner workings of our command-line-based file system implemented in C++. We'll delve into the design choices, the underlying data structures, and the decision-making process that shaped IMFS. Additionally, you'll find clear instructions on setting up the development environment.

System Design

Classes and Components

FileSystem

At the heart of IMFS is the **`FileSystem`** class. It acts as the orchestrator, handling file and directory operations, user commands, and system state management.

Methods

- **`mkdir(const string& dirName)`**: Create a new directory.
- **`cd(const string& path)`**: Change the current directory.
- **`ls() const`**: List the contents of the current directory.
- **`cat(const string& fileName) const`**: Display the contents of a file.
- **`touch(const string& fileName)`**: Create a new file.
- **`echo(const string& text, const string& fileName)`**: Write text to a file, replacing its content.
- **`mv(const string& source, const string& destination)`**: Move a file or directory to another location.
- **`cp(const string& source, const string& destination)`**: Copy a file to another location.
- **`rm(const string& target)`**: Remove a file or directory.
- **`run()`**: Start the file system and take user input.
- **`saveState(const string& filePath) const`**: Save the file system state to a JSON file.
- **`loadState(const string& filePath)`**: Load the file system state from a JSON file.

Node

The **`Node`** structure represents a file or directory in IMFS, holding essential information such as name, type, content, and children.

Properties

- **`name`**: The name of the file or directory.
- **`isDirectory`**: A boolean indicating whether it's a directory.
- **`content`**: The content of the file (relevant for files only).
- **`children`**: A vector of unique pointers to child nodes.

JSON Serialization

IMFS uses the **`json`** class from the nlohmann/json library to serialize and deserialize the file system state to and from JSON.

Command Processing
User commands are processed through the **`processCommand`** method, providing a modular and extensible approach to manage commands.

Design Decisions
- **Node Structure:** The nested **`Node`** structure simplifies the representation of the file system's hierarchical structure.
- **Smart Pointers:** Leveraging **`std::unique_ptr`** for child node management ensures robust memory handling and ownership.
- **Command Processing:** The central **`processCommand`** method enhances modularity, making it easy to manage and expand commands.
- **JSON Serialization:** Serializing the file system state to JSON enables seamless persistence and reloading of the system.

Setup Script
Let's get your development environment set up for IMFS. Follow these steps:

- **Clone the Repository:**

```
git clone https://github.com/your/repo.git

cd your_project_directory
```

- **Download Google Test:**

```
git clone https://github.com/google/googletest.git

cd googletest

mkdir build

cd build

cmake ..

Make
```

Build the File System:
```
cd your_project_directory

mkdir build

cd build

cmake ..

Make
```

Run Tests:

```
./run_tests
```

Execute the File System:

```
./file_system_executable
```

This documentation aims to provide a human-friendly insight into IMFS, making your journey through the codebase smooth.