Design Document for MemFS

Project Overview

The MemFS project is a basic in-memory file system simulator designed in C++ that allows users to interact with files through various commands such as create, write, delete, read, exists, and ls (list files). This system is multi-threaded, providing thread-safe operations through mutexes and atomic operations. The main goal of this system is to simulate file management functionalities with safe concurrent access, latency simulation, and command-based interaction.

Class and Method Summary

1. FileMetadata

A struct that stores metadata about each file, including:

- string content: The content of the file.
- size_t size: Size of the file content.
- string creationDate: The date when the file was created.
- string lastModified: The date when the file was last modified.

2. MemFS

The primary class that manages the in-memory file system, containing:

- **Data Members**:
  - unordered_map<string, FileMetadata> files: Stores files and their metadata.
  - mutex fsMutex: Ensures thread-safe access to shared resources.
  - atomic<int> createdCount: Counts successful file creations atomically for concurrent operations.
- **Helper Functions**:
  - string getCurrentDate() const: Returns the current date as a string.
  - void createFile(const string &filename): Creates a new file in a thread-safe way.
  - void writeFile(const string &filename, const string &data): Writes content to a file in a thread-safe way.
  - void deleteFile(const string &filename): Deletes a file in a thread-safe way.

- **Public Functions**:
  - void createFiles(int numFiles, const vector<string> &filenames): Creates multiple files concurrently.
  - void writeToFile(int numFiles, const vector<pair<string, string>> &fileData): Writes data to multiple files concurrently.
  - void deleteFiles(int numFiles, const vector<string> &filenames): Deletes multiple files concurrently.
  - void readFile(const string &filename) const: Reads and displays the content of a file.
  - void listFiles(bool detailed = false) const: Lists files, with an option to include detailed metadata.
  - void executeCommand(const string &command): Parses and executes commands from user input.

Core Functionalities

1. File Creation (createFiles and createFile)

- **Purpose**: To create files in the in-memory system.
- **Concurrency**: Multiple threads are spawned to create files concurrently. Thread safety is ensured using a mutex to protect shared data (files map).
- **Error Handling**: If a file already exists, an error message is displayed. If multiple files are created, a success message is displayed.

2. File Writing (writeToFile and writeFile)

- **Purpose**: To write data to files in the in-memory system.
- **Concurrency**: Uses multiple threads to write data concurrently to various files, with a mutex to prevent race conditions.
- **Error Handling**: If a file does not exist, an error message is displayed. If data is successfully written, a success message is displayed.

3. File Deletion (deleteFiles and deleteFile)

- **Purpose**: Deletes files from the in-memory system.
- **Concurrency**: Deletes multiple files concurrently with thread safety via a mutex.
- **Output**: If files do not exist, a message lists non-existent files; if files are successfully deleted, a success message is displayed.

4. File Reading (readFile)

- **Purpose**: Reads and displays the content of a specified file.

- **Usage**: Displays file content if it exists; otherwise, shows an error.

5. Listing Files (listFiles)

- **Purpose**: Lists all files in the system, with an option for detailed metadata.
- **Output**: If -l is specified, lists each file's size, creation date, last modified date, and name. Without -l, only filenames are shown.

6. Command Execution (executeCommand)

- **Purpose**: Parses and executes commands issued by the user in the format command [options].
- **Commands Supported**:
    - create: Creates one or more files.
    - write: Writes data to one or more files.
    - delete: Deletes one or more files.
    - read: Reads content from a specified file.
    - ls: Lists all files with optional metadata.
    - exit: Exits the program.

Design Considerations

1. **Concurrency**: File operations (create, write, delete) are designed to be thread-safe. Mutexes prevent race conditions by ensuring that only one thread can access or modify shared resources at any given time.
2. **Atomic Operations**: createdCount uses std::atomic to ensure thread-safe updates for file creation counters.
3. **Error Handling**: Each file operation includes basic error handling, such as checking for file existence before operations and displaying error messages when files are not found.
4. **User Commands**: Commands are parsed and executed through executeCommand(), supporting a user-friendly interface for file management.

Future Enhancements

1. **File Locking**: Implement per-file locking to allow finer control over individual file operations.
2. **Logging System**: Add a logging mechanism to track and log file operations and errors.
3. **Persistent Storage**: Extend MemFS to store data in a physical file to make it persistent across sessions.
4. **File Permissions**: Add file permissions to restrict read/write access to files.

5. **Improved Error Reporting**: Enhance error messages for better user feedback.

Example Usage

```
memfs> create file1.txt
File created successfully

memfs> create -n 4 file1.txt file2.txt file3.txt file4.txt
error: another file with same name exists
Files created successfully

memfs> write file1.txt “Hello, world!”
successfully written to file1.txt

memfs> write -n 2 file2.txt “Hi my first name is Krishna” file3.txt “My surname is Biswakarma”
successfully written to the given files

memfs> read file1.txt
Hello, world!

memfs> delete -n 1 file4.txt
File deleted successfully
```

[NOTE - Here, we are using the `-n` flag when deleting a single file to demonstrate the optional behaviour for using the flag when dealing with a single file.]

```
memfs> ls
file2.txt
file3.txt
file1.txt

memfs> ls -l
size   created      last modified   filename
27     14/11/2024   14/11/2024      file2.txt
24     14/11/2024   14/11/2024      file3.txt
13     14/11/2024   14/11/2024      file1.txt

memfs> exit
exiting memFS
```

Conclusion

The MemFS class provides a thread-safe, in-memory file system that supports basic file operations in a concurrent environment. It allows users to manage files interactively through a command-line interface, demonstrating concurrency control, atomic operations, and error handling. This design is extensible, supporting future enhancements to incorporate more sophisticated file system features.