OS Lab2

File Systems

—

Felix Cenusa
Yamen Chams
BTH Software Engineers

—

18-12-2024

Report on File System Implementation

1. Introduction

The purpose of this lab was to create a file system that supports common operations like creating, deleting, moving, and copying files and directories. The system also enforces permissions and manages disk space. This report explains how we built the system, the decisions we made along the way, and how it handles different tasks.

---

2. System Design

2.1 Overview

The file system uses a FAT (File Allocation Table) to manage storage blocks and directory tables to organize files and folders. The root directory (/) is the base of the hierarchy, and the system supports subdirectories and basic file operations. Disk space is split into fixed-size blocks, and these are allocated or freed as needed.

2.2 Key Components

- **Directory Entries (dir_entry)**: Store file / folder metadata (name, size, type, etc.).
- **FAT**: Keeps track of which blocks are free, in use, or the end of a file.
- **Directory Tables**: The root directory holds all top-level files, while the current directory table tracks where the user is working, that being if the user went in said folder.

2.3 Disk Management

The system reads and writes blocks to a simulated disk using the Disk class. This is done to ensure that the program can run anywhere and be showcased. Each block is a fixed size, which simplifies how data is stored and accessed.

---

3. Implementations

3.1 Formatting the Disk

The format function sets up the system by clearing the FAT and directory tables. It initializes all blocks as free (except for the root and FAT blocks) so the disk is ready for use and zeros out all the data on the disk.

3.2 File Operations

- **create**: Adds a new file, allocating disk blocks and linking them in the FAT if the data doesn't fit in one block. File content is stored in these blocks, and metadata is added to the directory table of the root or current directory table if we are in a folder.
- **cat**: Reads a file's content and prints it, checking for read permissions first.
- **ls**: Lists the contents of the current directory, including names, sizes, types, and permissions.

3.3 Directory Management

- **mkdir**: Creates a new directory, initializing its table with .. entries that point to the parent directory / folder. Works with relative or absolute paths.
- **cd**: Changes the current directory by updating the directory table to match the target location. Can use local or absolute paths.
- **pwd**: Prints the full path of the current directory by traversing back to the root temporarily and printing what we traversed through in a formatted manner.

3.4 Copying and Moving Files

- **cp**: Copy file to folder, meaning make a file with the same contents in that folder. Copy file to another file, meaning copy contents and change name: Reads the source file, allocates new blocks, and writes the copied data. Metadata like size and permissions is duplicated.
- **mv**: Renames or moves a file.

3.5 Permissions

- **chmod**: Updates a file or directory's permissions (read, write, execute).
- **Checks**: Operations like cat and append ensure the required permissions are met.

---

4. Data Structures

4.1 FAT

The FAT maps each block to the next block in a file or marks it as free (FAT_FREE) or the end of a file (FAT_EOF).

4.2 Directory Tables

Directories use tables of dir_entry structs to organize their contents. Each entry holds a file or folder's metadata, such as its name, size, and permissions.

4.3 dir_entry

Stores:

- file_name: The file or directory's name.
- size: The file's size in bytes (directories don't have sizes).
- first_blk: The block where the file or directory starts.
- type: Indicates if it's a file or directory.
- access_rights: Permissions for read, write, and execute.

---

5. Design Decisions

- **Using FAT**: It's simple to implement and makes it easy to manage blocks for files.
- **Directory Tables**: Splitting root and current directory tables keeps navigation clean and avoids unnecessary complexity.
- **Permissions**: Adding access rights makes the system more realistic and avoids unwanted operations.

---

6. Error Handling

- **Duplicate Names**: Checks ensure no two files or directories in the same folder share a name, we had it as a feature before but it was not wanted so we took it away.
- **Invalid Paths**: Commands like cd and rm validate paths before acting.
- **Permission Checks**: Errors are returned if an operation violates permissions.
- **Non-Empty Directories**: Directories can only be deleted if they're empty.
- **A lot more :)**

---

7. Challenges

- **Directory Navigation**: Managing paths (.. , / , and nested directories) was tricky. We used parent (..) to handle it cleanly.
- **Disk Space**: Finding enough free blocks was challenging, so we implemented checkSpaceInDisk function to make it easier.
- **Error Cases**: Handling edge cases like moving files to the same location required additional validation logic.
- **A lot more as well (:**

---

8. Testing

8.1 Test Scripts

We ran the provided test scripts (test_scriptX.cpp)(0 > X >= 5), which checked operations like creating files, copying directories, and enforcing permissions.

8.2 Manual Testing

We manually tested additional scenarios, including:

- Moving files across directories.
- Appending files with restricted permissions.
- Handling invalid commands or paths.

---

9. Conclusion

The file system supports all required features, including hierarchical directories, file operations, and permissions. It passed the provided tests and works as expected in manual scenarios. While the implementation meets the assignment’s goals, future improvements could include expanding a directory table if it's filled, allowing users to press the up and down arrows to go to previous commands, and having the username or disk name as the text when starting the program instead of filesystem>. Overall it was a great learning experience and once we got going with it implementing more and more things got easier once we got used to how the entire system worked.