Marvellous Custom Virtual File System (CVFS)

Project Overview

This project is a comprehensive Custom Virtual File System (CVFS) implemented entirely in Java. It simulates the fundamental structure and operations of a file system kernel, including memory management concepts like the Super Block, Inode List, and User File Descriptor Table (UFDT). The system is designed to run an interactive shell that accepts common Linux-like file commands, such as creat, unlink, read, write, ls, and stat.

Author: Shardul Tapkire

Date: 04/10/2025

File Details

File Name	Description
CVFS.java	A single Java file containing all necessary classes (MarvellousConstants, SuperBlock, Inode, UAREA, etc.) to implement the simulated in-memory file system and its command-line shell.

Features and Data Structures

The CVFS operates using several interconnected data structures that simulate a real-world file system layout.

1. File System Metadata Structures

Class	Description	Key Members	Role in CVFS
MarvellousConstants	Defines system limits and error codes.	MAXINODE (5), MAXFILESIZE (100), MAXOPENEDFILES (20), READ (1), WRITE (2).	System configuration and robust error handling.

Class	Description	Key Members	Role in CVFS
BootBlock	Placeholder for boot information.	Information	Conceptual element in the simulated boot process.
SuperBlock	Global file system statistics.	TotalInodes, FreeInodes	Tracks the availability of file creation slots.

---

2. Core File Management Structures

Class	Description	Key Members	Underlying Structure
Inode	**In-memory representation of a file's metadata and data.** Collectively forms the **DILB** (Disk Inode List Block).	FileName, InodeNumber, ActualFileSize, Permission, Buffer (the actual file data - char[]).	**Singly Linked List (head)** to manage file slots.
FileTable	Stores dynamic context for an **opened** file.	ReadOffset, WriteOffset, Mode, ptrinode (reference to the associated Inode).	Element of the UFDT.
UAREA	Represents the Process Control Block (PCB).	ProcessName, **UFDT** (User File Descriptor Table): Array of FileTable references.	Maps the **File Descriptor (FD)** index to the specific opened file's context.

---

3. Shell Commands and Core Functionality

The MarvellousCVFS class provides the implementation for the following shell commands:

Command	Usage	Description	Key Function(s)
creat	creat <FileName> <Permission>	Creates a new file, allocates an Inode and a FileTable entry (returns FD).	CreateFile()
unlink	unlink <FileName>	Deletes the file by resetting the Inode and freeing the FileTable slot (if open).	UnlinkFile()
ls	ls	Lists the names of all files currently created in the file system.	ls_file()
stat	stat <FileName>	Displays metadata: name, size, link count, type, and permissions.	stat_file(), IsFileExists()
write	write <FileDescriptor> (prompts for data)	Writes user-input data to the file, updating the WriteOffset and ActualFileSize.	write_file()
read	read <FileDescriptor> <Size>	Reads specified Size of data from the file, starting from the ReadOffset.	read_file()
help	help	Displays a list of all supported commands.	DisplayHelp()

Command	Usage	Description	Key Function(s)
man	man <command>	Displays the detailed manual page for a specific command.	ManPage()
exit	exit	Gracefully terminates the CVFS shell.	MainShell() loop termination

---

4. Technical Implementation Details

Initialization (StartAuxilaryDataInitialisation)

The system initializes in a fixed, in-memory state:

- A linked list of **5** Inodes (MAXINODE) is created, representing the total file capacity.
- The Super Block is set with 5 FreeInodes.
- The UFDT is prepared to hold up to **20** opened file contexts (MAXOPENEDFILES).

File Allocation

- **File Descriptor (FD):** The FD is the array index in the UFDT (0 to 19). This index is returned to the user upon successful file creation.
- **Write/Read Operations:** Both operations use separate, independently managed offsets (WriteOffset, ReadOffset) within the FileTable. This simulates sequential read/write access.
- **Data Storage:** File content is stored in a fixed-size char[] Buffer within the Inode, limited by MAXFILESIZE (100 bytes).

Error Handling

The system uses a set of negative integer constants (ERR_...) to return specific failure reasons, simulating system calls' return values.

---

5. Future Enhancements

The current implementation provides a solid foundation for simulating core file system principles. Potential future work and significant enhancements include:

A. File System Structure Improvements

Enhancement Area	Feature Details	Impact
Dynamic Inode List	Replace the fixed-size Inode linked list with a dynamic structure (e.g., a LinkedList or ArrayList in Java) that can grow up to a much larger (though still fixed) limit, separate from the immediate boot block.	Allows the file system to scale and supports more realistic file capacity.
Data Block Simulation	Implement **indirect addressing**. Replace the single char[] Buffer inside the Inode with an array of pointers to separate, smaller data blocks.	Allows file sizes to exceed the size of a single data block, making file size limit (MAXFILESIZE) adjustable and files larger than 100 bytes.
Directory Structure	Implement a simple **Directory structure**. Create a special Inode type (e.g., FileType = 3 for Directory) whose data buffer stores a list of file names and their corresponding InodeNumbers.	Enables file path support (e.g., creat /home/user/file.txt), not just flat file names.

B. Command and Functionality Expansion

Enhancement Area	Feature Details	New Commands/Methods
File Operations	Implement the ability to open existing files, close files, and change offsets.	open <FileName> <Mode>, close <FileDescriptor>, lseek <FileDescriptor> <Offset> <Whence> (using START, CURRENT, END constants).
Permissions	Implement a more granular permission system (e.g., separate R/W/X for Owner/Group/Other) and enforce chmod and chown logic.	chmod <FileName> <NewPermission>
Linking	Implement hard links, which share the same Inode (incrementing LinkCount).	link <OldName> <NewName>

C. Robustness and Usability

Enhancement Area	Feature Details	Impact
Persistence	Add functionality to save the entire state of the MarvellousCVFS (SuperBlock, Inode list, Buffers) to an actual file on the host OS (.img file) and load it upon startup.	Makes the file system non-volatile; data persists across program executions.
Multi-Threading	Implement a simple multi-threaded environment where multiple simulated processes (UAREAs) can concurrently try to open and access files.	Introduces challenges like concurrency control and resource locking, crucial in OS design.

Enhancement Area	Feature Details	Impact
Code Refactoring	Encapsulate the command processing logic (parsing, error mapping) into a dedicated Shell class to improve modularity and separation of concerns from the core file system logic.	Improves code cleanliness, testing capabilities, and maintenance.

---

6. Learning Outcomes

By completing and exploring this project, I have gained valuable experience in:

- **OS Concepts:** Understanding the internal mapping mechanisms between User Mode (FD) and Kernel Mode (Inode/FileTable).
- **Data Structuring:** Using interconnected structures (Linked List of Inodes, Array of FileTables) to represent complex relationships.
- **System Programming:** Implementing robust error-checking and status-code return mechanisms typical of system calls.
- **Code Design:** Applying OOP principles to model real-world OS components like the Super Block and Inodes.
- **Input Handling:** Developing logic to parse multi-parameter commands and handle input streams in a shell environment.