# Write –Up Deliverable

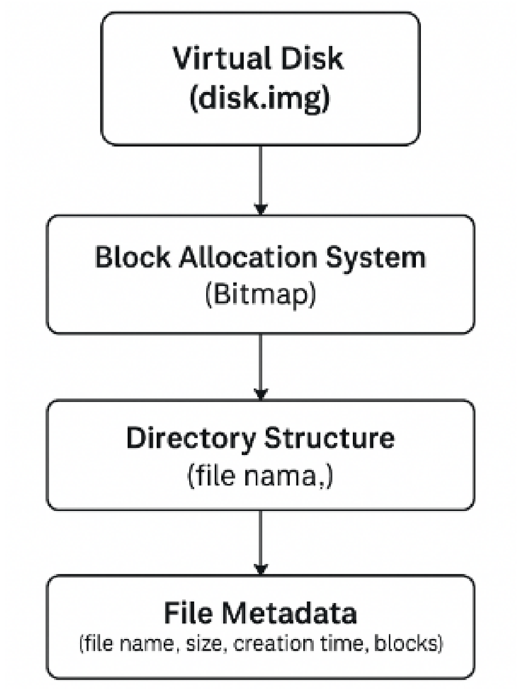
Baby Vennela Kothakonda

**Table of Contents**

## 1. Introduction

This is a project aimed at designing and implementing a custom file system (FS) that carries out several features such as disk level storage, file navigation, and file handling. The overall idea of this file system is inspired from basic file systems like FAT, ext, Btrfs, but still provides simplicity, flexibility, and customization for use in educational purposes. The mechanics including creation, deletion, directory navigation and more advanced such as block allocation, file metadata and multi-block files support are woven into the system.

## 2. Architecture and Structure


Virtual Disk (disk.img)
Block Allocation System (Bitmap)
Directory Structure (file nama,)
File Metadata (file name, size, creation time, blocks)

The components of our custom file system architecture are divided into the following:

1. **Virtual Disk (disk.img)**: A virtual disk file that is used to simulate real disk storage. The disk is divided into blocks of fixed size (Siron, 2016). This file is used as the storage medium of files and directories.

2. **Block Allocation System**: A bitmap keeps a record of free and used blocks on the disk, and keeps track of whether its block is allocated. For space efficiency and ease block management, it supports multi block files (Karresand et al., 2019).

3. **Directory Structure**; the directory structure is hierarchical (e.g., /home/user/file.txt). A directory can contain either just files or other directories. The directory structure is based on a simple tree based structure for easy navigation.

4. **File Metadata**: Each file in the system has its file metadata and it includes file name, size, creation time and which blocks he occupies. The directory structure or the linked list of inodes contains metadata (Ganger et al., 2000).

## 3. Design Decisions

- **Choice of Bitmap for Block Allocation**: Bit map uses bits to represent the disk blocks and '1' represents the used blocks and '0' for free blocks. It makes efficient use of free space for block management.

- **Choice of Directory Structure**: A directory is represented by a file structure with nodes, i.e. children and directories. It is a hierarchical organization of files to enable easy directory traversal and management.

- **File Operations**: The file system successfully handles the following operations:

    - **Create**: Files are created and initialized with empty content.

    - **Read**: The contents of a file can be read block by block.

    - **Delete**: Files are deleted, and their blocks are marked as free in the bitmap.

    - **Write**: Data is written to files, and new blocks are allocated if needed.

    - **Touch**: Creates an empty file with the specified name.

Because the data structures (bitmap for block allocation, tree-based directory structure, and simple linked list for file metadata) are simple and efficient in an educational context, it was decided to use them. Ease of implementation here has been balanced with adequate functionality.

## 4. Comparison to Other File Systems

- **FAT File System**: The FAT file system is an old type of file system that implemented the linked list of blocks to manage files. It works perfectly fine but is fragmented and inefficient in dealing with large files (Rane & Singh, 2024). Finally, our custom file system outperforms FAT through the use of a bitmap for the block allocation, more space efficient and easier to manage.

- **ext File System**: Ext file system (for example ext3 and ext4) has been made advanced with journaling and inode based storage (Fairbanks, 2012). Although our system does not journal, it provides relatively simple block allocation and directory navigation for education.

- **Btrfs File System**: Btrfs is a very advanced modern file system with things like copy on write and snapshots (Dakic et al., 2024). Our custom file system provides basic functionality for the file system learning as well.

## 5. Prototype Implementation

The prototypes of the custom file system consist of:

1. **Filesystem Core Logic**: This is implemented in filesystem.py file. Block allocation, file metadata maintenance and file operation handling like creates, reads, writes, deletes, and so on are all handled by it.

2. **Disk Driver**: The disk.py file is simulating the physical disk (disk.img). It has functions for reading and writing blocks to the disk image file.

3. **Shell Interface**: One can interact with the file system through the shell.py file which is a command line interface. This supports mkdir, cd, ls, pwd, create, read and delete commands.

### 5.1. Code Prototype

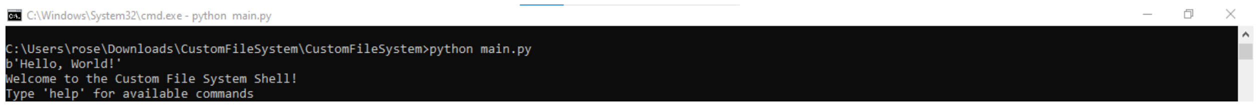
❖ **filesystem.py**: Core logic for block allocation, file creation, deletion, and reading.

❖ **disk.py**: Disk simulation with read/write operations.

❖ **shell.py**: Command-line interface to interact with the file system.

## 5.2 Functionality Demonstration:

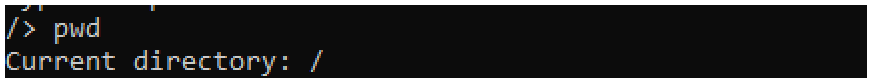
1. **Initialize the Disk**: Using the command python main.py to initialize the disk.



```
C:\Users\rose\Downloads\CustomFileSystem\CustomFileSystem>python main.py
b'Hello, World!'
Welcome to the Custom File System Shell!
Type 'help' for available commands
```

2. **File Operations**: Applying the shell commands to interact with the file system:

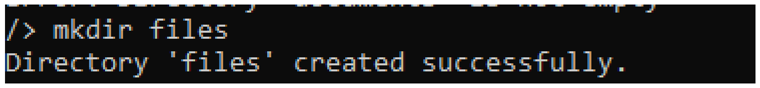
   - Pwd: to check where we are in directory



```
/> pwd
Current directory: /
```

   - mkdir <directory_name>: Creating a new directory.

     Example: mkdir files



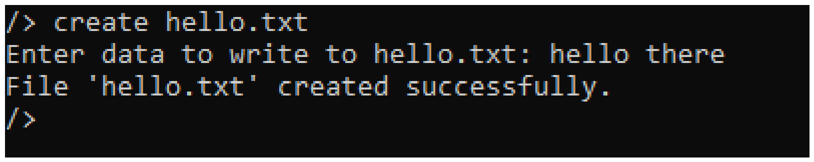
```
/> mkdir files
Directory 'files' created successfully.
```

   - cd <directory_name>: Changing the current directory.

     Example: cd files

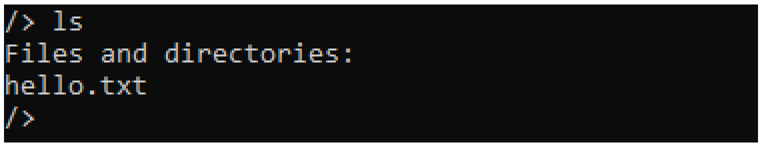
   - create <file_name>: Create a new file.

     Example: create hello.txt



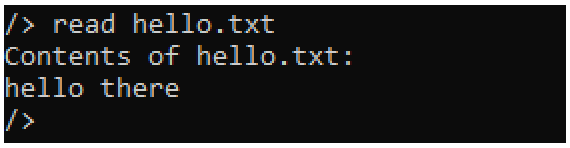
```
/> create hello.txt
Enter data to write to hello.txt: hello there
File 'hello.txt' created successfully.
/>
```

   - ls: I will give a list of the files and directories in the current directory.


```
/> ls
Files and directories:
hello.txt
/>
```

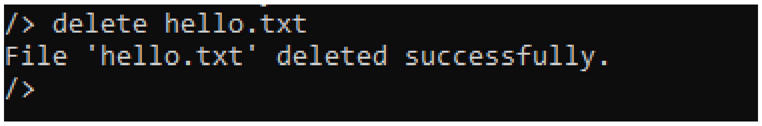
- read <file_name>: Reading the contents of a file.

  Example: read hello.txt

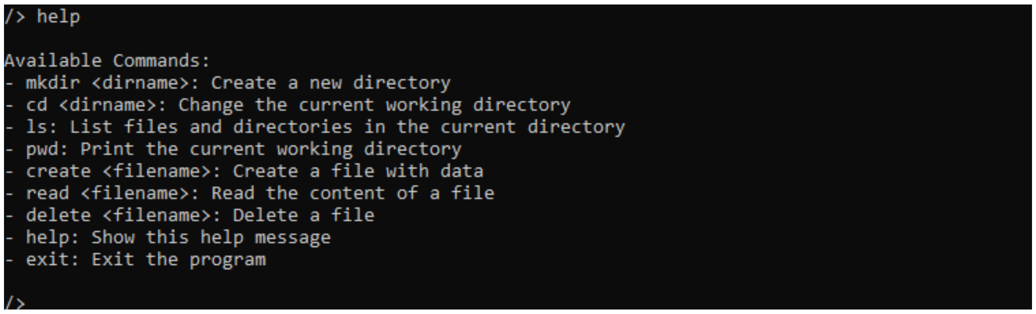

```
/> read hello.txt
Contents of hello.txt:
hello there
/>
```

- pwd: Printing the current working directory.
- delete <file_name>: Deleting a file

  Example: delete documents


```
/> delete hello.txt
File 'hello.txt' deleted successfully.
/>
```

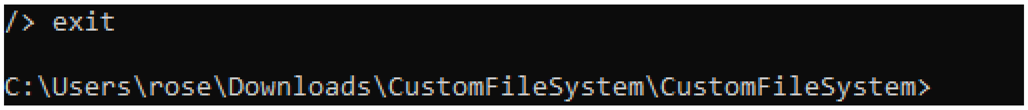
- help: Displaying a list of available commands and their descriptions.


```
/> help

Available Commands:
- mkdir <dirname>: Create a new directory
- cd <dirname>: Change the current working directory
- ls: List files and directories in the current directory
- pwd: Print the current working directory
- create <filename>: Create a file with data
- read <filename>: Read the content of a file
- delete <filename>: Delete a file
- help: Show this help message
- exit: Exit the program

/>
```

- exit: Exiting the shell and stops the program.


```
/> exit

C:\Users\rose\Downloads\CustomFileSystem\CustomFileSystem>
```

## 7. Timeline & Milestones

| **Milestone** | **Deliverable** | **Due Date** | **Description** |
|---|---|---|---|
| Milestone 1 | Design Document | Week 1 | Submit a design document outlining the file system structure, chosen data structures, and key functionalities. |
| Milestone 2 | Core Functionality Implementation | Week 2 | Submit the initial code implementation with file creation, deletion, and basic directory management. |
| Milestone 3 | Advanced Functionality and Testing | Week 3 | Submit the full code with features like ls, pwd, cd, file reading, and writing. |
| Milestone 4 | Final Code Submission & Documentation | Week 4 | Submit the final implementation with complete documentation, shell interface, and a working prototype. |
| Final Submissio n | Full Report and Demo Presentation | Week 5 | Submit the final report, prototype, and present a demo of the system. |

## 8. Conclusion

The file system we designed and implemented in this project offers the basic file operation such as creation, deletion, and navigation, efficient block allocation as well as metadata manipulation. We illustrated how we simplified the operation of the file system by comparing it to existing file systems such as FAT, ext, and Btrfs. Future improvements could include journaling, file permissions, persistence to make the system more robust and so on. The value of the insights this project gives to file system design, management, and optimization is considerable.

## References

Dakic, V., Kovac, M., & Videc, I. (2024). High-Performance Computing Storage Performance and Design Patterns—Btrfs and ZFS Performance for Different Use Cases. *Computers*, *13*(6), 139. https://doi.org/10.3390/computers13060139

Fairbanks, K. D. (2012). An analysis of Ext4 for digital forensics. *Digital Investigation*, *9*, S118–S130. https://doi.org/10.1016/j.diin.2012.05.010

Ganger, G., Soules, C., Patt, Y., & Soules, C. (2000). Soft Updates: A Solution to the Metadata Update Problem in File Systems. *ACM Transactions on Computer Systems*, *18*(2), 127–153. https://pdos.csail.mit.edu/archive/6.097/readings/journal-softupdate.pdf

Karresand, M., Axelsson, S., & Dyrkolbotn, G. O. (2019). Disk Cluster Allocation Behavior in Windows and NTFS. *Mobile Networks and Applications*, *25*(1), 248–258. https://doi.org/10.1007/s11036-019-01441-1

Rane, R., & Singh, A. (2024, April 15). *Demystifying File Systems: A Comprehensive Exploration of Data Organization*. https://doi.org/10.13140/RG.2.2.31160.35845

Siron, E. (2016, August 4). *Understanding and Working with VHD(X) Files*. Altaro DOJO | Hyper-V. https://www.altaro.com/hyper-v/understanding-working-vhdx-files/