CLUSTER INNOVATION CENTRE,

UNIVERSITY OF DELHI

END SEMESTER PROJECT REPORT

FILE SYSTEM IMPLEMENTATION IN LINUX

Dheeraj sonekar(12018) | Diksha Tripathi (12019) | Dushant Bansal(12020) | Harsh Kumar(12021) | Harshvir Sandhu (12022) | Ishan Bansal(12023) | Ishika Rai (12024) | Jatin(12025) | Jayshree Choudhary (12026) | Madhav (12028) | Mehak Sood (12029) | Mohit Kumar (12030) | Naman Priyadarshi (12031) | Nikunj Saini(12032)

ACKNOWLEDGMENT

It was a great pleasure for us to present our work on this Project report as a part of the fourth semester's curriculum in B.Tech (Information Technology and Mathematical Innovations) at the Cluster Innovation Centre, University of Delhi. We are very grateful to our mentor, Professor Anjani Kumar who with his support and venerated guidance rendered this project an adequate success. Getting the opportunity to perform such activities at such an organization is an exquisite learning experience that made a mark at the profoundest part of our mind.

ABSTRACT

Linux file system has a hierarchical file structure as it contains a root directory and its subdirectories. All other directories can be accessed from the root directory. A partition usually has only one file system, but it may have more than one file system.

A file system is designed in a way so that it can manage and provide space for non-volatile storage data. All file systems required a namespace that is a naming and organizational methodology. The namespace defines the naming process, length of the file name, or a subset of characters that can be used for the file name. It also defines the logical structure of files on a memory segment, such as the use of directories for organizing the specific files. Once a namespace is described, a Metadata description must be defined for that particular file.

TABLE OF CONTENTS

INTRODUCTION

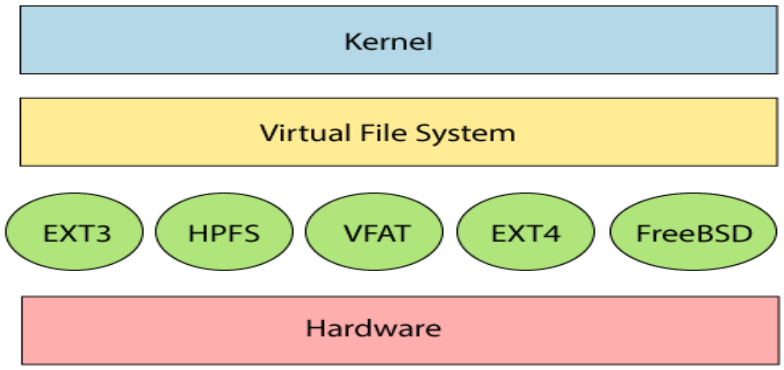
A Linux file system is generally a built-in layer of a Linux operating system used to handle the data management of the storage. It helps to arrange the file on the disk storage. It manages the file name, file size, creation date, and much more information about a file.

A file system is designed in a way so that it can manage and provide space for non-volatile storage data. All file systems required a namespace that is a naming and organizational methodology. The namespace defines the naming process, length of the file name, or a subset of characters that can be used for the file name. It also defines the logical structure of files on a memory segment, such as the use of directories for organizing the specific files. Once a namespace is described, a Metadata description must be defined for that particular file.

The data structure needs to support a hierarchical directory structure; this structure is used to describe the available and used disk space for a particular block. It also has the other details about the files such as file size, date & time of creation, update, and last modified. Also, it stores advanced information about the section of the disk, such as partitions and volumes.

The advanced data and the structures that it represents contain the information about the file system stored on the drive; it is distinct and independent of the file system metadata.

Linux file system contains two-part file system software implementation architecture.



FUNCTION LIST

1. **format()**

   This function does not need to be called explicitly by the user in terminal. This function runs as soon as the File management system is executed. This function intialises the root directory of the system in the image of the disk. This root directory will be the parent of the sub directories that the user will make explicitly.
   This function also generates the meta-data about the root directory.

2. **cd**

   This function is one the most basic and important functions of a file management system. Through this function user can easily traverse between directories. The following is their syntax
   - To go in a child directory-
     cd [child directory name]
   - To go back to previous directory-
     cd..

3. **ls**

   This function lists all the files that are present in the current directory. There is no specified syntax for this command. We just type **ls,** and it gives us the names of all the files present in that directory.

4. **mkdir()**

   This function id responsible for creating child dirctories form the current directory. This function can be called by using the keyword **mkdir**. The following is the syntax for this command in our file management system-
   mkdir [directory name]

5. **rmdir()**
   Just like mkdir makes the child directory, the function of rmdir is to remove the empty child directories created intentionally/unintentionally. If the directory that we want to remove is empty, this command is executed successfully. If the directory has file, irrespective if the size of the file i.e. empty or filled, it will give us error and terminate itself. This command is executable to al the directories except the root directory. The syntax is given below-
   rmdir [directory name]

6. **create()**
   Once we reach to our destination directory, we need to create a file to store the data. Using this command, we can create empty files in the current directory. The file type that we want to create has to given in the name itself. Following is the syntax for create with the example of a text file
   create [file name].txt // .txt is the extension for text files.

7. **rm()**
   Just like we had function to remove directory, this function helps in deletion of unwanted files in the working directory. The rm() function explicitly removes the file name specified. The syntax is given below
   rm [file name].txt

8. **open()**
   This function allows us to read and write data in the file created. The file that is created using create command is in accessible to write or read function by default. So these files need to be opened before performing any operation in the files. Given below is the syntax for running this command in the system-
   open [file name].txt

9. **close()**

   An open file can not be saved by the system. In order to save the data fed in the file, we need to close the file. This function is responsible for closing the file write operations. The syntax for this command is as follows-

   quit

10. **write()**

    Once the file is created and is set open we can write the data in the file using the write command. The write function can be carried out in 3 ways i.e.

    - append- add data to the end of the data existing in the file.
    - truncate- erase the data upto a specific lenght and then add the new data from there.
    - overwrite- completely delete the data in the file and replace it with the new data

    Given below is the syntax for write function

    write [fd number]
    a // write type - a→ append, t→ truncate, o→ overwrite
    Hello // data
    quit // close the file

11. **read()**

    The data that we feed into the file is only accessible through the program itself. So to get the data in readable format, we use this read command. For this command, we need to open the file first, then we can perform read function-

    open [file name].txt
    read [fd number] //fd number comes from the FAT32

12. **sf()**

    This function is to save the file and their contents according the their FAT id's that we obtain from the FAT32. The output of this function gives the list of all directories and the files stored at a particular id. There is no argument for this function, we simply write **sf**

13. **exit_sys()**

    This is the last function that we have created. Once we are done creating files, reading and writing the data from the files, we want to safely exit the file manager. So we write this command to exit the program

    exit_sys

PROJECT EXPERIENCE

Initially, we explored the different types of file system structures. We gathered information about the FAT32 type file structure and decided to base our implementation of the file system on it. While we initially explored Unix File Structures based on inodes, we decided to base our implementation of the File System on FAT32 Type File Structure.

FAT (File Allocation Table) file system was developed in the late 1970s and early 1980s and was the file system supported by the Microsoft® MS-DOS® operating system. While in comparison to other systems, the performance of FAT is poor as it uses simple data structures, making file operations time-consuming and inefficient disk space utilization in situations where many small files are present but for the same simple design and legacy it is supported by almost all existing operating systems for personal computers. This makes it a useful format for solid-state memory cards and a convenient way to share data between operating systems. Conventional FAT32 file system which can address large storage media and is supported by all major desktop operating systems, is still the most widely used file system in portable digital devices.

At the beginning of the project, we quickly realized that we would need to make a set of private functions which would do all of the primitive low level operations of the file system. We regularly scheduled online meets to decide upon the public functions through which the user would be able to access the file system via our terminal. These have been specified in the Function List. After finalizing the public functions, we distributed the work among ourselves as follows:

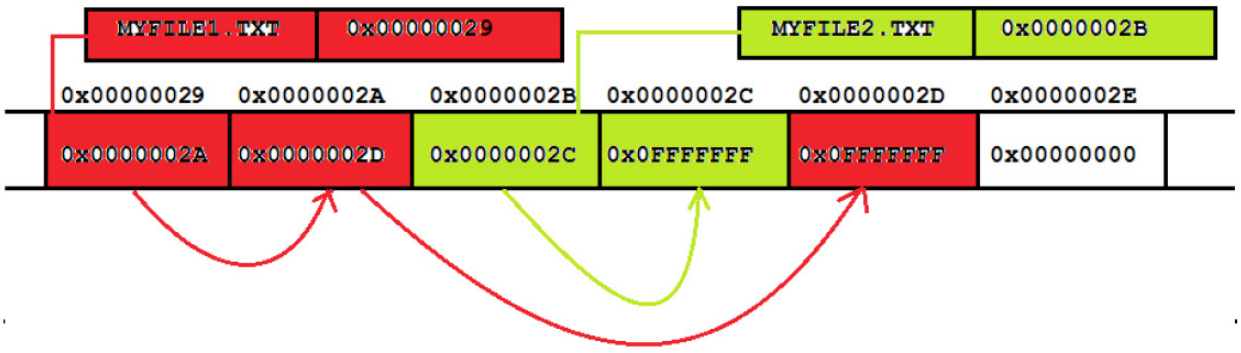
Names		Functions
Ishan	Mehak	Utils
Dushant	Mohit	Basic
Madhav	Dheeraj	Read
Harsh	Jatin	Write
Ishika	Diksha	Delete
Nikunj	Harshvir	Create
Jayshree	Naman	Others

We worked on the above function groups in pairs. After completion, we integrated all the functions into a single file.

During the course of the project, we delved into the concepts of FAT32 Structure. The following table shows the order of the data structures that compose a FAT32 disk volume.

Boot Sector	Reserved Sector	FAT (Copy 1)	FAT (Copy 2)	File & Directory Sectors

We also thoroughly understood the FAT Data Structure for FAT 32 Volume.



Snapshot of FAT data structure for FAT32 Volume

After understanding, we programmed some functions which would make the process of reading and writing these various file system constructs to and from the disk. We worked sequentially, first building the data block and FCB retrieval functions, then moving onto working with directories (cd, rmdir, mkdir, etc.). Atlast we worked on our File functions (read, write, create, open, etc.). Working sequentially gave us a clear mind-map of the implementation strategy which would assist us throughout development of our project.

CONCLUSION

In this project we were able to design a C based File Management System capable of creating, modifying and deleting a file through the terminal. Our program can make child directories as and when required and is capable of handling upto 10 files in one go. The following are the key features of our program that we have made-

1. Creates root directory, child directory and files.
2. We can write data in files through terminal in three different modes, i.e. append, truncate and overwrite.
3. Through this program we can traverse through different directories, create and delete files as and when required.
4. We have created functions that can delete the whole directory that is empty. If the directory has any file, it will show an error.

BIBLIOGRAPHY

- https://www.geeksforgeeks.org/inode-in-operating-system/
- https://en.wikipedia.org/wiki/File_system
- https://www.youtube.com/watch?v=n2AAhiujAqs
- https://github.com/cadesalaberry/SimpleFS
- https://github.com/YvesChan/FileSystem
- https://github.com/sysprog21/simplefs
- https://www.youtube.com/watch?v=tMVj22EWg6A