BasicOS — File system

Handed on friday 2nd of December 2022

Group:
Ahmed Ghaleb — `ahmed.ghaleb@eurecom.fr`
Guillaume Ung — `guillaume.ung@eurecom.fr`
William Chieu — `william.chieu@eurecom.fr`

EURECOM
Sophia Antipolis

Contents

1 Introduction

In the context of the BasicOS course taught by Pr. Ludovic Apvrille at EURECOM we had to design a basic file system[1]. The feature of this file system is that it handles a no space left on device error.
What is a file system? A file system is a method and data structure that the operating system uses to control how data is stored and retrieved.

Our program has been written in C and has a set of commands to manipulate files and the file system (`create, write, read, ls, size, remove`). All the data is stored in a file on the host computer.

The usage instructions are in written in the file `README.md`.

2 General approach

To design our file system we have decided to have one **superblock** and a fixed number of **inodes**. We thought that it would make it simpler for us.

Table 1: Structure

Superblock Fixed #	**Inodes** Fixed #	**Datablocks** Variable #
16B	10 000 * 104B	x * 512B

The superblock is a C structure containing fields like; **`int size, int dbcount`**.
The inodes are C structures containing information about the file like; **`char filename, int inodenumber, timestamps`**.
The datablocks are placed after the inodes in the file system, and we decided for the sake of simplicity that we would not have fragmentation. All the datablocks linked to a file are contiguous.
Functions for each command are written in separate files. **`src/myfs.c`** was mainly used for argument parsing and function calling.
We also had to code a set of utility functions to facilitate and avoid code duplication.

Function repartition

- Guillaume: **`read, remove`**
- William: **`create, ls`**
- Ahmed: **`write, size`**

[1]cf. `https://perso.telecom-paristech.fr/apvrille/BasicOS/project.html`

3 Utility functions

Here is a list of the functions which were used to make the program. Their implementation can be found under `src/definitions.c`.

```
inode_t get_inode(); // given a filename, returns the corresponding
    inode
int get_free_inode(); // given a table, returns index of free inode
int get_free_db(); // returns ptr to a given # of contiguous datablocks
int load_inodes(); // loads inodes of the FS in a data structure
int myfs_load(); // loads the FS into data structures
int myfs_init(); // budget version of create, used for tests
```

Utility functions

4 Main commands

4.1 create

Create takes as input the size of the file system so I decided to fix the number of inodes and to make the number of datablocks a variable to fill the file system with datablocks, thus creating a file system of the right size. I initialized the inodes and the datablocks by allocating the space necessary for every inodes and datablocks. Afterwards, I filled the inodes and the datablocks by writing the null character in order to create a file system of the right size. There is a minimum size that can be given by the user since the file system is defined with a fixed amount of inodes so the size given by the user has to be enough to create at least 1 datablock.

4.2 write

This function writes a file already present on the host computer to the file system given to the command. For this, if the source file size is not greater than 5MB we store its content in a buffer. Then we look into the file system to find free datablocks and inode, if free, we initialize an inode with information about the source file and write at a free datablock pointer returned by the function **`get free inode();`**. If there is no more space on the file system or no inode, the program will prompt the user asking whether they want to delete old files. If yes, oldest files will be deleted until enough space is freed. If no, the program will cleanly exit.

4.3 ls

ls takes as input the directory, the function read the given directory and while there is still a file to read, it keeps printing the file name. Whenever, a directory is encountered, the program detects it with S_ISREG and reads its contents by recursing. At the end, it exits the directories thanks to **`chdir("..")`** to return to the parent directories so ls can print the rest of the files in the previous directories.

4.4 read

The read function reads the data from the input file. We first check if the file exist by looking at the inode and if it exists, we read each character from the datablock linked to the file and add them to a string to finally, print that string.

4.5 size

For size, function that lists size of a given directory, since we did not make inodes for directories, we thought that it would list the files with a filename containing the desired folder name using the **`strstr();`** function. Then we just print the amount of characters in bytes, kilobytes, megabytes or gigabytes. For the `'-stat'` option, partition table size is the sum of every character from index `0` to `inode-size*inode-count` and lastly for the size of the file partition we count the characters starting from the first datablock pointer to the end of the file system.

4.6 remove

The remove function removes the file from the filesystem. To do so , it first checks if the file has an inode and if not, then the file doesn't exist. Then, it seeks the datablocks in the filesystem linked to the file and empties them (i.e each character becomes '0'). It does the same for the inode data.

5 Conclusion

Overall, we found the project quite challenging because we had to get acquainted with how a file system works. The hardest part being implementing what we had imagined in C, a new (not to Ahmed fortunately) and very rigorous language. In that sense, it was all the more pleasing to see the program compile (with no warnings) and work correctly. There were many aspects we wanted to fine-tune like, optimization, error handling, security but learning C on the fly took us a lot of time and energy.