La Salle 2020

The Shooter

Advanced Operating Systems

Marc Llort Maulion

10-5-2020

index

Requirements

The goal of the practice *The Shooter* is to understand in depth the volumes of file systems formatted in EXT and FAT. In our case, we will focus on the EXT2 and FAT16 versions.

To achieve this knowledge, it will be necessary to be able to carry out different operations, where each one will be a phase of the practice, with the images of these volumes EXT2 and FAT16:

- Info (/ info)
- Find simple (/ find)
- Find depth (/ find)
- Delete (/ delete)

Info

In this functionality, we will have to detect what type is the volume that the user enters us, and in case it is EXT2 it is necessary to show:

INODE INFO: Inode Size, Num Inodes, First Inode, Group Inodes, Free Inodes

BLOCK INFO: Block Size, Reserved Blocks, Free Blocks, Total Blocks, First Block, Group Blocks, Group Frags

VOLUME INFO Volume name, Last check, Last edit, Last write

In the case of a FAT16 file:

Filesystem, System Name, Sector Size, Sectors By Cluster, Reserved Sectors, Number of FATs, MaxRootEntries, Sectors by FAT, Label

You must also check that the number of parameters is correct, the file / volume exists ...

In this case an example would be: *./ shooter / info testFat*

Find

For phase 2 and 3, the find functionality must be implemented. As can be deduced from the name, this feature is about finding a file or directory within an EXT2 or FAT16 volume. The difference between phase 2 and 3 will be that in the first, we will need to be able to find the file within the " *root directory* ", Therefore the root of the file system.

In the second case, the program will have the ability to find this file or directory both at the root of the file system and within other folders.

Once we find the file, we will display a message indicating its size. If it is a directory, we will simply indicate that we have found the folder.

You must also check that the number of parameters is correct, the file / volume exists ...

In this case an example would be: *./ shooter / find testFat file*

Delete

The last phase of the practice, where it is necessary to carry out the deletion of a binary file at the root of the file system. The file must be completely deleted from both EXT2 and FAT16.

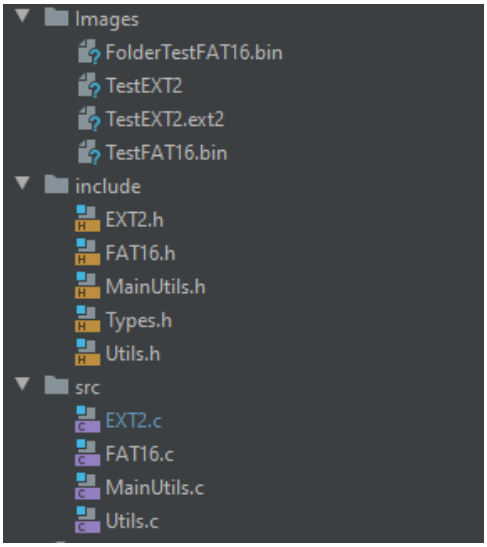
You must also check that the number of parameters is correct, the file / volume exists ...

In this case an example would be: ./ *shooter / delete testFat file*

When programming, you need to do it in C, and use a makefile while compiling the program.

Design

As for the structure followed to perform the practice, I decided to fragment the different specific functions of each file system into a .ci .h for EXT2 and FAT16.



On the other hand, I have a Utils file, for those functions that I use in both file systems, such as custom strlen functions, capitalize a word ...

Finally, I have a MainUtils file, which contains the functions that main.c uses to call the functionalities of each file system. The 3 functions are:

- infoFileSystem: receives as a parameter the file system, and from this, identifies what type it is (EXT2 or FAT16), and once identified, initializes different variables of the identified file system, and executes the function of *printInfo* that touches.

- searchInFileSystem: receives as a parameter the file system and the name of the file to be found, and from these, identifies what type it is (EXT2 or FAT16), and once identified, initializes different variables of the identified file system, and executes the function of *findFile* that touches.

- deleteInFileSystem: receives as a parameter the file system and the name of the file to be deleted, and from these, identifies what type it is (EXT2 or FAT16), and once identified, initializes different variables of the identified file system, and executes the function of *deleteFile* that touches.

In the main.c file, it simply makes a switch according to the function they introduce us and reports the different errors that the command entered by CLI may have.

If there is no error in the order formatting, the MainUtils function is called, which handles the Handling error in case the file system is not recognized.

Next, I will explain in detail the functions of both EXT2.c and FAT16.c.

EXT2

In this file are all the specific functions of EXT2 to perform the functionalities required by the program.

In the respective .h, there are the amounts needed to work, along with the function declaration and some global constants and variables (fd, filename).

The functions of EXT2.c are as follows:

- *Ext2Volume getInfoEXT2 ()*
  Use the fd (global variable), to read the volume, scrolling with *lseek* to the different bits of the volume to find the necessary information, and store it at the type *Ext2Volume (* explained later in data structure).

- *struct tm * getTime (uint32_t time)*
  Function used in *printInfoExt2,* to correctly print information about when the last changes were made.

- *int isEXT2 (int fileDescriptor)*
  From the fd, it is placed at position 1116 and reads 32bits, to know if it is an EXT2 file or not.

- *Ext2Directory getInfoEXT2Directory (int fd, unsigned long filePosition)*
  From the fd and the position received, it returns information about the type of file that exists, its name, size ... Once I find a file, I use this function to be able to print its size. *Ext2Directory* is explained later in data structure.

- *InodeEntry getInodeData (int fd, Ext2Volume ext2, unsigned int inodeNum)*
  With the fd, volume information and inode number, I read and store the selected inode information. *InodeEntry* is explained later in data structure.

- *void printInfoExt2 (Ext2Volume ext2)*
  From the information collected by *getInfoEXT2* displays the information explained above in the requirements.

- *int findFileEXT2 (char * fileName)*
  "General" file or directory search function. Gather all the information needed for the function *findFileExtVolume* and then in case of file found, it is in charge to detect the type of file or directory, and to print its information.

- *unsigned long findFileExtVolume (int fd, Ext2Volume ext2, char * fileName, unsigned char * fileType, unsigned char * rootDir, int inodeNumber)*
  Function responsible for iterating the entire volume to find the file that has the same name as *filename.* It is called recursively to be able to search within subdirectories of the root of the file system. The variable *fileType,* serves to indicate to the function *findFileEXT2* what type of file was found. *InodeNumber* indicates the inode number from which to search. *RootDir,* indicates whether the file was found at the root of the system or not, useful for reusing the function while deleting.

- *int deleteFileEXT2 (char * fileName, char * fileSystem)*
  "General" file deletion function. Gather all the information needed for the function *findFileExtVolume* and then in case of file found, it is in charge to detect the type of file or directory, and to call the function *deleteFileEXT2Volume*.

- *int deleteFileEXT2Volume (int fd, unsigned long filePosition)*
  From the receiving position and the fd, it is responsible for completely deleting the volume file.

Info

At the same time performing the info functionality is simply about going to file system positions and reading them. To position the cursor we use *lseek*. Once positioned it is necessary to do *read* with a size of the exact size we want to read, together passing the pointer of the variable where we want to store this information, in this case it would be EXT2Volume.

The section on data structure explains these positions together with the functionality of each variable.

Find

To search for a file or directory in the EXT2 file system, keep in mind that the files are organized in inodes.

At first, you have to look at the superblock (position 1024). This will be the starting point of our search. From here, we will have to move from node to node (128 bytes).

We find that the inode has 12 direct pointers, 1 indirect, 1 double indirect and one triple indirect. When we look at an inode, if we find that a block has a value of 0, it means that it is empty, so it does not link to any other block.

Once we read a block of information, we check if it is the same file name as the one we are looking for. In case it is not the name, but it is a directory, we will do a recursive search inside the directory, re-calculating the position of the inode from adding from the Inode table 128 bytes N-1 strokes.

Delete

At first, as with find, we need to find the position of the file in the file system. To perform the search, use the function *find* explained above.

Once the file is found, both the information block needs to be deleted.

To do this, override the directory (ext2dir) where the file inode is located.

FAT16

In this file are all the specific functions of FAT16 to carry out the functionalities required by the program.

In the respective .h, there are the amounts needed to function, along with the function declaration and some global constants and variables (fd).

The functions of FAT16.c are as follows:

- *FAT16Volume getInfoFAT16 ()*
  Use the fd (global variable), to read the volume, scrolling with *lseek* to the different bits of the volume to find the necessary information, and store it at the type *FAT16Volume (* explained later in data structure).

- *int isFAT16 (int fileDescriptor)*
  From the fd, it is placed at position 54 and reads 8bits, to know if it is a file of type FAT16 or not.

- *FAT16Directory getInfoFAT16Directory (int fd, unsigned long filePosition)*
  From the fd and the position received, it returns information about the type of file that exists, its name, size ... Once I find a file, I use this function to be able to print its size. *FAT16Directory* is explained later in data structure.

- *void printInfo FAT16 (FAT16Volume fat16)*

  From the information collected by *getInfoFAT16* displays the information explained above in the requirements.

- *int findFileFAT16 (char * fileName)*
  "General" file or directory search function. Gather all the information needed for the function *findFileFAT16Volume* and then in case of file found, it is in charge to detect the type of file or directory, and to print its information.

- *unsigned long findFileExtVolume (int fd, FAT16Volume fat16, char * fileName, unsigned char * fileType, unsigned char * rootDir, uint32_t firstCluster)*

  Function responsible for iterating the entire volume to find the file that has the same name as *filename*. It is called recursively to be able to search within subdirectories of the root of the file system. The variable *fileType,* serves to indicate to the function *findFileFAT16* what type of file was found. *firstCluster* indicates the cluster number from which to search. *RootDir,* indicates whether the file was found at the root of the system or not, useful for reusing the function while deleting.

- *int deleteFileFAT16 (char * fileName, char * fileSystem)*
  "General" file deletion function. Gather all the information needed for the function *findFileFAT16Volume* and then in case of file found, it is in charge to detect the type of file or directory, and to call the function *deleteFileFAT16Volume.*

- *int deleteFileFAT16Volume (int fd, unsigned long filePosition)*
  From the receiving position and the fd, it is responsible for completely deleting the volume file.

Info

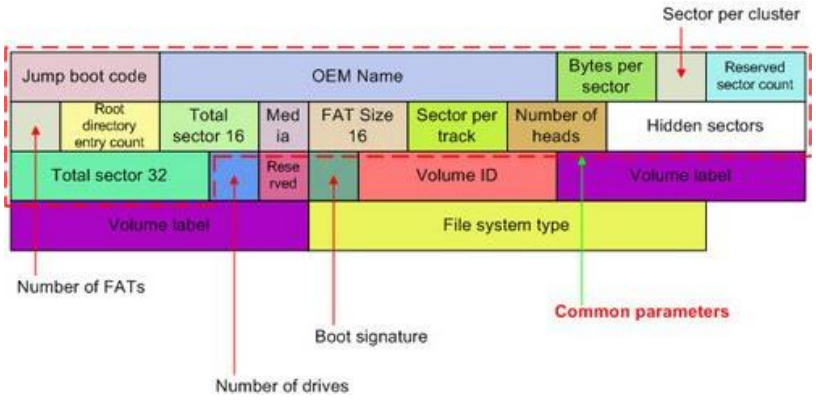
At the same time performing the info functionality is simply about going to file system positions and reading them. To position the cursor we use *lseek*. Once positioned it is necessary to do *read*
with a size of the exact size we want to read, together passing the pointer of the variable where we want to store this information, in this case it would be FAT16Volume.

The section on data structure explains these positions together with the functionality of each variable.

Find

To start a search on the FAT16 file system, you need to position the pointer in the RootDirectory, which is located after a reserved region, and a FAT region.

Once in rootDirectory, each entry occupies 32 bytes, structured as follows:



Therefore, it is necessary to iterate through the rootDirectory looking to see if the file name matches (given that the names are in uppercase). If you find a folder, you need to make a recursive call to the search function, where the address is from the firstCluster as follows:

```
uint32_t firstSectorOfCluster = ((firstCluster - 2) * fat16.sectorCluster) + fat16.reservedSectors + (fat16.numberFats * fat16.sectorsFat)
+ (((fat16.rootEntries * 32) + (fat16.sectorSize - 1)) / fat16.sectorSize);

firstSectorOfCluster = firstSectorOfCluster * fat16.sectorSize; lseek (fd, firstSectorOfCluster, SEEK_SET);
```

Once we find the file or directory, applying the “& 0x20” mask to the fileAttributes of the file, we will know if it is a file, or if 0x10 is a directory.

Delete

As we do in EXT2, we must first find the position of the entry where the file is located. Once found, iterate through the entire information block to 0.

Data structures

All the types used throughout the programming of the program, are stored in the file *types.h.*

EXT2

First of all, while performing the functionality of printing the basic information of the EXT2 file system, we use the type *EXT2Volume:*

```
typedef struct {
    // Inode Info uint32_t inodesCount; uint32_t
    freeInodesCount; uint32_t inodesGroup;
    uint32_t inodeSize; uint32_t firstInode;

    // Block Info uint32_t blockCount; uint32_t
    freeBlockCount; uint32_t reservedBlocksCount;
    uint32_t firstDataBlock; uint32_t blocksGroup;
    uint32_t blockSize; uint32_t fragsGroup;

    // Volume Info char volumeName
   [16]; uint32_t lastCheck; uint32_t
   lastMounted; uint32_t lastWritten; }
   Ext2Volume;
```

We can see that in this type all the information it stores is stored in 32-bit (4-byte) int's, except for the volume name, which we store in an array of char's.

Here is a short explanation of what each variable holds:

inodesCount: Position 1024, saves the total number of inodes on the partition.
freeInodesCount: Position 1068, saves the number of free inodes on the partition.
inodesGroup: Position 1064, saves the number of inodes per group.
inodeSize: Position 1112, keeps the size of a node.
firstInode: Position 1108, the link to the first node that can be used to save files.
blockCount: Position 1032, saves the total number of blocks in the partition.
freeBlockCount: Position 1040, saves the number of free blocks of the partition.
reservedBlocksCount: Position 1036, saves the number of reserved blocks in the partition.
firstDataBlock: Position 1076, saves position of the first block with information.
blocksGroup: Position 1056, saves the number of blocks per group.
blockSize: Position 1052, save the size of each block.
fragsGroup: Position 1060, saves the total number of fragments in each group.
volumeName: Position 1144, save the volume number.
lastCheck: Position 1088, saves the last time the partition was checked.
lastWritten: Position 1072, saves the last time it was written to the partition.

Second, the Ext2Directory structure, where it stores information once we find a file or directory. It gives us information such as its node, its size, type and file name.

```
typedef struct {
        uint32_t inode; uint16_t recordLength;
       unsigned char nameLength; unsigned char
       fileType; char fileName [255]; } Ext2Directory;
```

And finally, the last structure used to store the information of each node is the following:

```
typedef struct {
        unsigned long i_mode; unsigned long i_uid;
        unsigned long i_size; unsigned long i_atime;
        unsigned long i_ctime; unsigned long i_mtime;
        unsigned long i_dtime; unsigned long i_gid;
        unsigned long i_links_count; unsigned long
        i_blocks; unsigned long i_flags; unsigned long
        i_osd1; unsigned long i_block [15]; unsigned long
        i_generation; unsigned long i_file_acl; unsigned
        long i_dir_acl; unsigned long i_faddr; unsigned
        long i_osd2; } InodeEntry;
```

In this case, we store all the information from the inode, although not all of it is necessary for the development of the practice.

FAT16

First of all, at the same time as performing the functionality of printing the basic information of the FAT16 file system, we use the type *FAT16Volume:*

```
typedef struct {
        char systemName [8]; uint16_t sectorSize;
       uint8_t sectorCluster; uint16_t
       reservedSectors; uint8_t numberFats;
       uint16_t rootEntries; uint32_t sectorsFat; char
       volumeName [12]; } FAT16Volume;
```

We can see that in this type all the information it stores is stored in different sizes, to suit each value we are looking for.

Here is a short explanation of what each variable holds:

systemName: Position 3, save the volume name.

sectorSize: Position 11, saves the number of bytes occupied by each sector.

sectorCluster: Position 13, keeps the number of clusters by sector.

reservedSectors: Position 14, saves the number of reserved sectors.

numberFats: Position 16, save the number of fat tables.

rootEntries: Position 17, saves the maximum number of directories at the root of the system.

sectorsFat: Position 22, saves the total number of sectors of the partition.

volumeName: Position 43, save tag (*label)* of the volume.

The second and last data structure used in the FAT16 system is:

```
typedef struct {
        char name [12]; char extension
        [4];
        unsigned char fileAttribute; uint16_t reserved;
       uint16_t time; uint16_t date; uint16_t firstCluster;
       uint32_t size; } FAT16Directory;
```

As in the case of EXT2, it stores the information once we find a file or directory. It gives us information such as the extension, its size, type and file name ...

Tests performed

Throughout the development of the practice I have used 4 different images, 2 of FAT16 and 2 of EXT2. Two of them are provided in eStudy, the other two generated by me.

Both of the generated images have folders and files inside them, so you can test the functionality of *find* in depth.

The structure of these generated images is as follows:

FolderTestFAT16.bin

main.o
folder
main.c

TestEXT2.ext2

Folder1
Folder11
text11.txt Folder12
text12.txt
Folder2
Folder21
text21.txt Folder22
text22.txt
Folder3
Folder31
text31.txt Folder32
text32.txt

Tests

```
marcllort@MAKINOTE:/mnt/c/Users/mac12/ShooterC$ ./shooter /find ./Images/TestEXT2.ext2 text32.txt
FILE POSITION: 34864 -- INODE: 27
File found! Size: 352 bytes
marcllort@MAKINOTE:/mnt/c/Users/mac12/ShooterC$ ./shooter /find ./Images/TestEXT2.ext2 carpeta32
Directory found!
marcllort@MAKINOTE:/mnt/c/Users/mac12/ShooterC$ ./shooter /info ./Images/TestEXT2.ext2
------ Filesystem Information ------

Filesystem: EXT2

INODE INFO
Inode Size: 128
Number of Inodes: 64
First Inode: 11
Inodes Group: 64
Free Inodes: 38

BLOCK INFO
Block Size: 1024
Reserved Blocks: 25
Free Blocks: 471
Total Blocks: 512
First Block: 1
Block Group: 8192
Frags Group: 8192

VOLUME INFO
Volume name: ext2>
Last check: Tue Apr 24 18:41:24 2007
Last mount: Tue Apr 24 19:33:11 2007
Last written: Tue Apr 24 19:33:16 2007
```

```
C:\Users\mac12\ShooterC\cmake-build-debug\ShooterC.exe /delete TestEXT2 com
File com has been deleted!
```

EXT2 tests

```
marcllort@MAKINOTE:/mnt/c/Users/mac12/ShooterC$ ./shooter /find ./Images/FolderTestFAT16.bin main.c
Undefined type of file MAIN    ! Size: 810 bytes
marcllort@MAKINOTE:/mnt/c/Users/mac12/ShooterC$ ./shooter /find ./Images/FolderTestFAT16.bin FOLDER
Directory found!
marcllort@MAKINOTE:/mnt/c/Users/mac12/ShooterC$ ./shooter /find ./Images/FolderTestFAT16.bin main.o
Undefined type of file MAIN    ! Size: 163464 bytes
marcllort@MAKINOTE:/mnt/c/Users/mac12/ShooterC$ ./shooter /info ./Images/FolderTestFAT16.bin
------ Filesystem Information ------

FileSystem: FAT16

System Name: WINIMAGE
Sector Size: 1024
Sectors per Cluster: 1
Reserved Sectors: 1
Number of FATs: 2
Maximum Root Entries: 512
Sectors per FAT: 16
Label: TEST2
```

```
C:\Users\mac12\ShooterC\cmake-build-debug\ShooterC.exe /delete FolderTestFAT16.bin main.o
File MAIN.O has been deleted!
```

FAT16 tests

```
marcllort@MAKINOTE:/mnt/c/Users/mac12/ShooterC$ ./shooter /info ./Images/FolderTestFAT16.binDDD
Error, filesystem not found! ./Images/FolderTestFAT16.binDDD
marcllort@MAKINOTE:/mnt/c/Users/mac12/ShooterC$ ./shooter /info ./Images/FolderTestFAT16.bin asdf
Incorrect number of parameters for the INFO function!
marcllort@MAKINOTE:/mnt/c/Users/mac12/ShooterC$ ./shooter /find ./Images/FolderTestFAT16.bin main.oasdsad
File not found!
```

Error handling tests

Observed Problems

Throughout practice I have come across several problems that have slowed me down quite a bit while developing.

The first and foremost, which has been repeated throughout practice, has been the ignorance of file systems, and not understanding exactly how each one worked. At first, as has happened to me many other times, I started programming without taking into account the understanding of the structure of each file system, and trying to learn about the system at the same time as I was programming.

This caused me many easily predictable errors, as once I stopped to understand how each of the file systems worked, everything went much smoother.

As for the programming, having already completed the subject of Operating Systems, and having very fresh C, along with the good programming practices that we had to apply to the practices of the subject, it has made it much easier for me to avoid problems. while making use of different more specific types for each data.

Regarding the program, the features that have caused me the most problems have been the search for files in depth (in both cases), and the deletion of files in EXT2.

In the case of the in-depth search, the problem was that the structuring of the code that had been performed in both cases was not the most appropriate to conduct a recursive in-depth search.

On the other hand, as I mentioned earlier, I also didn't have much knowledge of how the directory system worked in each of the file systems.

Second, with the removal of EXT2, I found that even though I knew the data structure of EXT2, as I had been working on it a lot in the earlier stages of the practice, I found that there was very little help and online examples to rely on, and mainly, not fully understanding what was the right way to completely delete a file, made me have to invest a lot of time to get to complete the operation of / *delete.*

Temporal Estimation

Due to the problems discussed above, and a major restructuring of the code to fix the *find* to be able to use it properly with phase 3, I estimate that I have invested a total of about 60 hours.

Conclusions

I find that it has been a practice that, although not its main goal, has helped me reaffirm the knowledge I already had of C, and put it into practice to learn from some file systems, which until now I was completely unaware of. .

Regarding file systems, it has helped me to understand what logical structure each of them follows, although I find that perhaps a little more help at the same time with the removal functions, would have saved me a lot of headaches, as there is very little information online.

Overall, this is a very comprehensive practice that makes you learn exactly how both EXT2 and FAT16 work.

Ratings

I think this way of learning is very interesting because in this way you learn to look for information and fix problems for yourself, although I think the number of hours you have to spend is very high, since having no guide to follow, you have to invest a lot of time in research.