



Sistemas Operativos y Redes II - 2^{do} semestre

2024

TP1 FileSystem FAT

Docente

- Benjamin Chuquimango

Integrantes

- Oviedo, Marcerlo
- Zavalla Gamarra, Martín Octavio

Fecha de entrega

- 18/09/2024

Índice

<u>Índice</u>	1
<u>Introducción</u>	2
<u>Desarrollo</u>	3
<u>1. Al Montarlo</u>	3
<u>2. Cargando el MBR</u>	3
<u>3. Cargando la tabla de archivos</u>	12
<u>4. Leyendo Archivos</u>	18
<u>Referencias</u>	25

Introducción

En el desarrollo de este trabajo, buscaremos analizar en profundidad la estructura y comportamiento de un sistema de archivos FAT12. Identificando el MBR (con sus tablas de particiones), el root directory, los archivos y directorios que éste contenga, y a su vez, realizaremos pruebas de lectura, creación y borrado de archivos. Para esto, contamos con un archivo **test.img**¹, que lo iremos leyendo a bajo nivel, ayudándonos con el uso del editor hexadecimal GHex, y utilizando programas en c.

Además, dejamos constancia de que, si se quiere ver la ejecución de los diversos programas trabajados en c, se pueden generar los ejecutables ingresando por línea de comando: **make** , y borrarlos con: **make clean** . Para esto, tuvimos que realizar modificaciones sobre el archivo **Makefile** proporcionado.

¹ Los **archivos binarios con la extensión .img**, almacenan imágenes de disco en bruto de los disquetes, discos duros o discos ópticos.

1. Al Montarlo

En primer lugar, hicimos la prueba para montar el archivo test.img, ingresando el comando: **mount test.img /mnt -o loop,umask=000**. El rol que cumple la colocación de **umask=000** es para deshabilitar permisos (lectura, escritura, ejecución) para todo usuario (root/propietario, usuarios que pertenecen al grupo, y demás usuarios del sistema que no pertenecen al grupo ni son propietarios). Cada valor de la tripleta especifica cuántos permisos se les sacará a cada tipo de usuario. En este caso, al ser los tres 0, no se quitarán permisos para archivos y directorios recién creados(por defecto los archivos tendrán permisos 666 y los directorios 777).

2. Cargando el MBR

En esta sección del trabajo estaremos poniendo el foco en la parte del MBR(Master Boot Record) del filesystem. Como primer tarea, mediante el editor GHex, debíamos mostrar los primeros bytes del MBR y sus diferentes tablas de partición. Para esto, también nos respaldamos con datos de nuestras fuentes de referencia.

Según la información que pudimos consultar, la estructura de datos de MBR ocupa 512 bytes, donde los primeros 446 se basan en un Boot Code que la BIOS ejecuta para arrancar el sistema operativo. Luego se tiene la tabla de 4 particiones(16 bytes por cada una), y un signature value(de 2 bytes) que es la firma del MBR para indicar su validez, cuyo valor siempre es 0xAA55.

Byte Range	Description	Essential
0-445	Boot Code	No
446-461	Partition Table Entry #1	Yes
462-477	Partition Table Entry #2	Yes
478-493	Partition Table Entry #3	Yes
494-509	Partition Table Entry #4	Yes
510-511	Signature value (0xAA55)	Yes

Estructura de datos del MBR

Conociendo los bytes donde se ubican cada una de las tablas de particiones, pasamos a buscarlas en el editor GHex, y mostramos los resultados en las siguientes imágenes:

Primera tabla de partición del byte 446(desde offset 0x1BE) al 461(hasta offset 0x1CD)

Segunda tabla de partición del byte 462(desde offset 0x1CE) al 477(hasta offset 0x1DD)

Tercera tabla de partición del byte 478(desde offset 0x1DE) al 493(hasta offset 0x1ED)

Cuarta tabla de partición del byte 494(desde offset 0x1EE) al 509(hasta offset 0x1FD)

Además, en la misma bibliografía, contamos con información por bytes, de lo que se encuentra en el Boot Sector, que nos servirá luego en el editor GHex, para ver qué datos encontraremos en sus primeros bytes:

Bytes	Purpose
0-2	Assembly code instructions to jump to boot code (mandatory in bootable partition)
3-10	OEM name in ASCII
11-12	Bytes per sector (512, 1024, 2048, or 4096)
13	Sectors per cluster (Must be a power of 2 and cluster size must be <=32 KB)
14-15	Size of reserved area, in sectors
16	Number of FATs (usually 2)
17-18	Maximum number of files in the root directory (FAT12/16; 0 for FAT32)
19-20	Number of sectors in the file system; if 2 B is not large enough, set to 0 and use 4 B value in bytes 32-35 below
21	Media type (0xf0=removable disk, 0xf8=fixed disk)
22-23	Size of each FAT, in sectors, for FAT12/16; 0 for FAT32
24-25	Sectors per track in storage device
26-27	Number of heads in storage device
28-31	Number of sectors before the start partition
32-35	Number of sectors in the file system; this field will be 0 if the 2B field above (bytes 19-20) is non-zero

primeros 36 bytes(del 0 al 35) del Boot Sector

Bytes	Purpose
0-35	(See previous table)
36	BIOS INT 13h (low level disk services) drive number
37	Not used
38	Extended boot signature to validate next three fields (0x29)
39-42	Volume serial number
43-53	Volume label, in ASCII
54-61	File system type level, in ASCII. (Generally "FAT", "FAT12", or "FAT16")
62-509	Not used
510-511	Signature value (0xaa55)

Información de los restantes bytes del Boot Sector

The screenshot shows the first 10 sectors of a file named 'test.img' in the GHex editor. The bytes are displayed in three columns: hex, ASCII, and decimal. The first three bytes (EB 3C 90) are highlighted in red, indicating they are part of the boot code. The byte at offset 0x9E (02) is highlighted in green, indicating it is a reserved sector indicator. Other bytes are highlighted in orange, purple, and blue. A status bar at the bottom shows 'Offset: 0x9E'.

Primeros bytes del MBR resaltados en el editor GHEx

Como podemos ver en la imagen dentro de GHEx, los primeros tres bytes remarcados en rojo corresponden a instrucciones en Assembler para hacer salto al boot code. Los siguientes 8 bytes son del OEM, que es un identificador que generalmente indica el nombre del sistema operativo o creador de la herramienta que formateó el sistema de archivos; en este caso, es el nombre de la herramienta *mkfs.fat*. Los 2 bytes que están marcados en naranja indican los bytes por cada sector(512). El recuadro en violeta indica los sectores por clúster(4). Los destacados en celeste muestran los sectores reservados(1). Y el byte en verde señala la cantidad de tablas FATs (2).

Luego, para leer todos los datos del MBR a través de un programa en c, contamos con el archivo adjunto **read_boot.c**, el cual ya tenía algunos campos agregados, y tuvimos que completarlo con los faltantes. Consultando la tabla de las imágenes de arriba, incorporamos los datos restantes que identificamos: **sectors per Cluster, reserved sectors, number of FATs , maximum numbers of files in the root directory, total sectors, media type, FAT size, sectors per track,**

number of heads, numbers of sectors before the start partition, total sectors, drive number, reserved(not used), boot signature.

```
marce998@marce-VM:~/Documentos/ejer_tp1_filesystem$ ./read_boot
Partition type: 1
Encontrado FAT12 0
Jump code: EB:3C:90
OEM code: [mkfs.fat]
sector_size: 512
Sectors per cluster: 4
Reserved sectors: 1
Number of FATs: 2
Maximum number of files in the Root Directory: 512
Total sectors : 2048
Media type: F8
FAT size : 2
Sectors per track: 32
Number of heads: 64
Number of sectors before the start partition: 0
Total sectors (large): 0
Drive number: 80
Boot signature: 29
volume_id: 0x06C8055F
Volume label: [NO NAME      ]
Filesystem type: [FAT12      ]
Boot sector signature: 0xAA55
```

*Datos del MBR mostrados al ejecutar el compilado de **read_boot.c***

Una vez visualizada toda la información del MBR, entramos en detalle sobre la primera tabla de partición de este sector, específicamente, buscamos saber si esa partición es booteable o no. Para encontrar en GHex el byte que nos dice si la primera partición es booteable, primero nos fijamos cómo es la estructura de las particiones del MBR, es decir, qué contienen los 16 bytes de cada una.

Byte Range	Description	Essential
0-0	Bootable Flag	No
1-3	Starting CHS Address	Yes
4-4	Partition Type	Yes
5-7	Ending CHS Address	Yes
8-11	Starting LBA Address	Yes
12-15	Size in Sectors	Yes

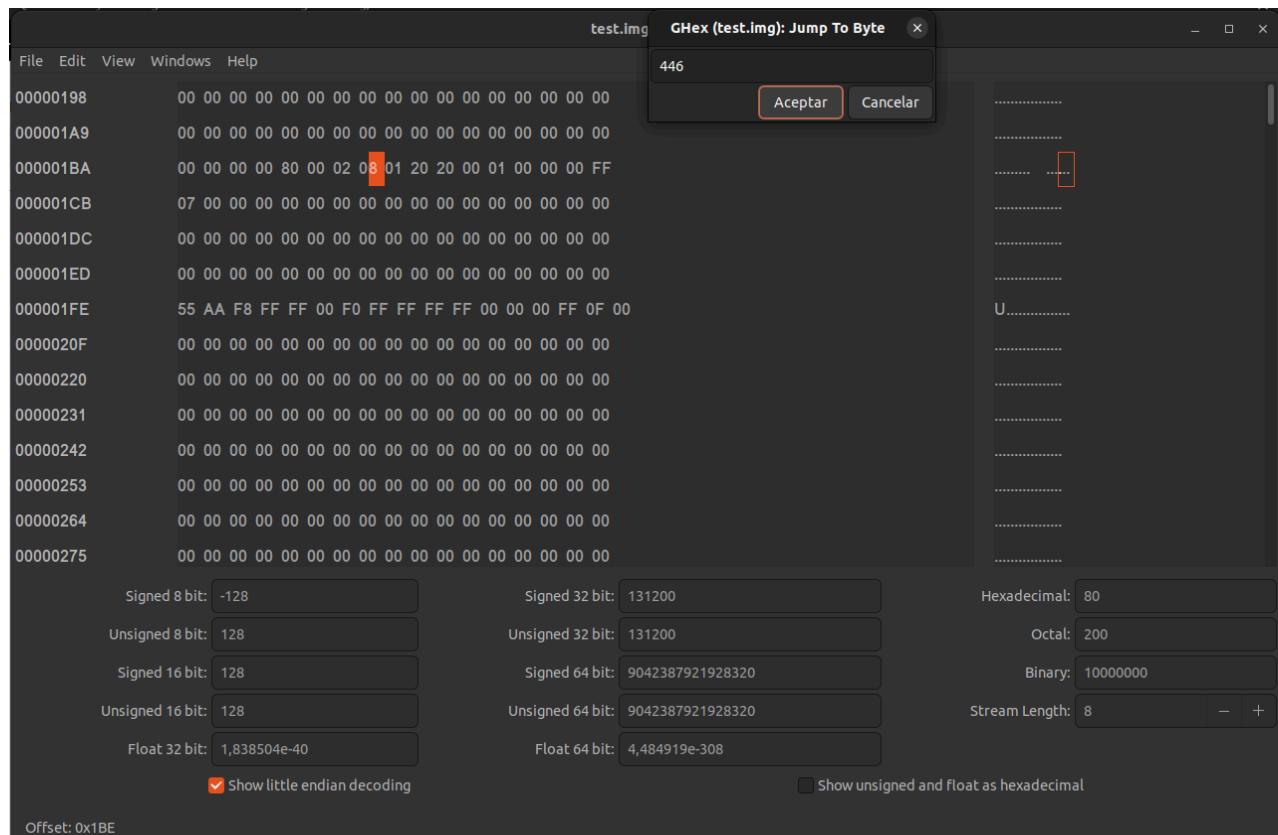
Estructura de cada partición del MBR

Como vemos en la tabla de arriba, el primer byte de cada partición, nos dice si es booteable o no. Para esto, el byte deberá indicar 80 en hexadecimal (dato sacado de una de nuestras fuentes de referencia) para decírnos que es booteable.

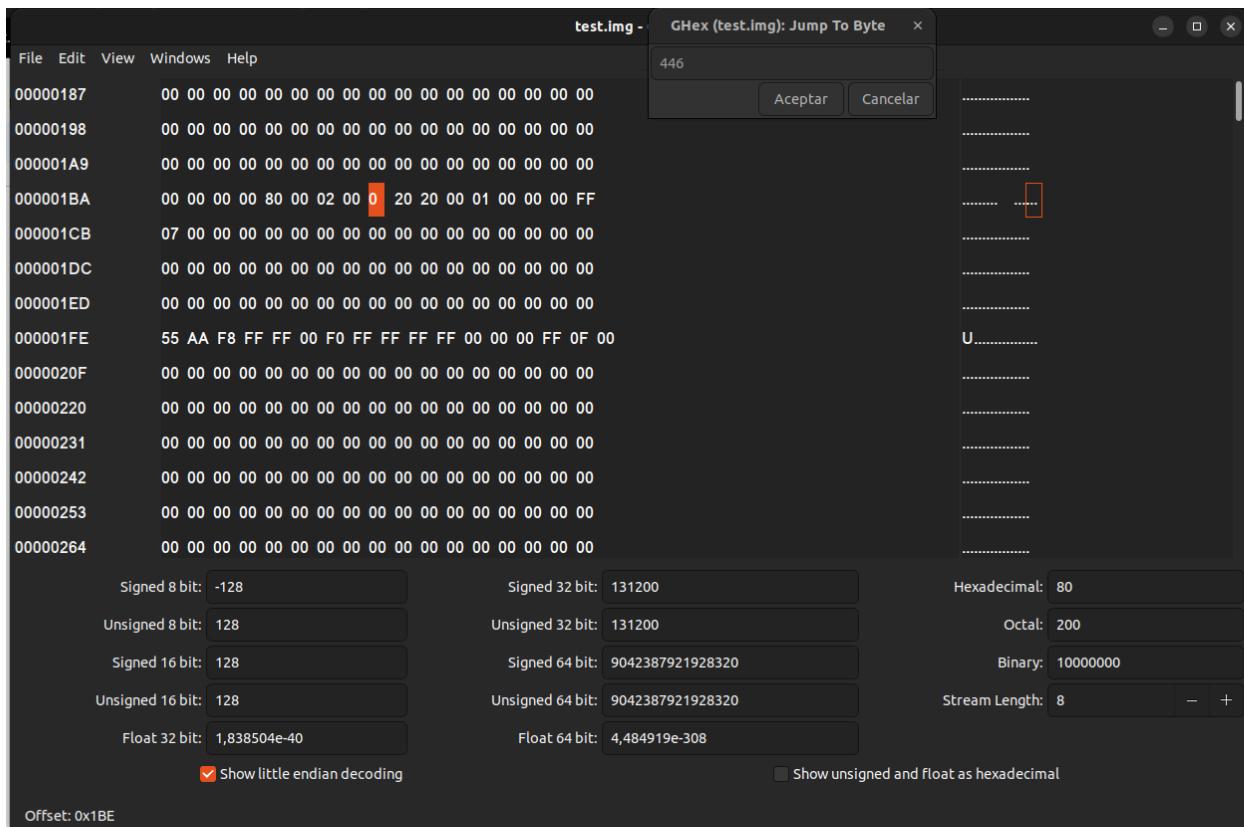
Offset (bytes)	Field length	Description
0x00	1 byte	Status or physical drive (bit 7 set is for active or bootable, old MBRs only accept 0x80, 0x00 means inactive, and 0x01–0x7F stand for invalid) ^[c]

Descripción del valor del primer byte de las particiones del MBR

Entonces, al saber que el primer byte de la primera partición es el 446, debemos fijarnos allí su valor.



8 leído de los primeros 4 bits del byte 446



O leído de los últimos 4 bits del byte 446

En las imágenes vemos que el byte 446(offset 0x1BE) tiene el valor hexadecimal 80, es decir, que la primera partición es booteable.

Además, mostraremos la información de esta primera partición, a partir de la ejecución del programa de **read_mbr.c**, en donde tuvimos que completar parte del código indicando el offset desde donde se debía leer la información, y además limitamos el ciclo for del código para que sólamente recorra la primera partición de las cuatro que existen. Como la primera partición comienza en el byte 446, su offset es **0x1BE** (el valor hexadecimal de 446 es 1BE).



Offset de donde comienza la primera entrada de partición de la estructura del MBR

```
marce998@marce-VM:~/Documentos/sor2/S0yR-2-Projects/FILESYSTEM$ ./read_mbr
Partition entry 1:
First byte 80
Comienzo de partición en CHS: 00:02:00
Partition type 0x01
Fin de partición en CHS: 00:20:20
Dirección LBA relativa 0x00000001, de tamaño en sectores 2047
```

Datos devueltos de la primera partición al ejecutar el compilado de read_mbr.c

3. Cargando la tabla de archivos

A continuación, nos enfocaremos en la parte del root directory del filesystem, para analizar los archivos y directorios existentes allí, y ver su comportamiento al crear y eliminar archivos. Antes de comenzar a ver los archivos desde el editor GHex y por la ejecución del programa en c, recurrimos de nuevo a la fuente de información proporcionada, para saber interpretar los datos de entrada que tiene éste sector en el FAT12:

FAT Root Directory Entry Format

Root Directory SFN Entry Data Structure	
Bytes	Purpose
0	First character of file name (ASCII) or allocation status (0x00=unallocated, 0xe5=deleted)
1-10	Characters 2-11 of the file name (ASCII); the "." is implied between bytes 7 and 8
11	File attributes (see File Attributes table)
12	Reserved
13	File creation time (in tenths of seconds)*
14-15	Creation time (hours, minutes, seconds)*
16-17	Creation date*
18-19	Access date*
20-21	High-order 2 bytes of address of first cluster (0 for FAT12/16)*
22-23	Modified time (hours, minutes, seconds)
24-25	Modified date
26-27	Low-order 2 bytes of address of first cluster
28-31	File size (0 for directories)

File Attributes	
Flag Value	Description
0000 0001 (0x01)	Read-only
0000 0010 (0x02)	Hidden file
0000 0100 (0x04)	System file
0000 1000 (0x08)	Volume label
0000 1111 (0x0f)	Long file name
0001 0000 (0x10)	Directory
0010 0000 (0x20)	Archive

* Bytes 13-22 are unused by DOS

Estructura de datos del Root Directory(32 bytes) de FAT

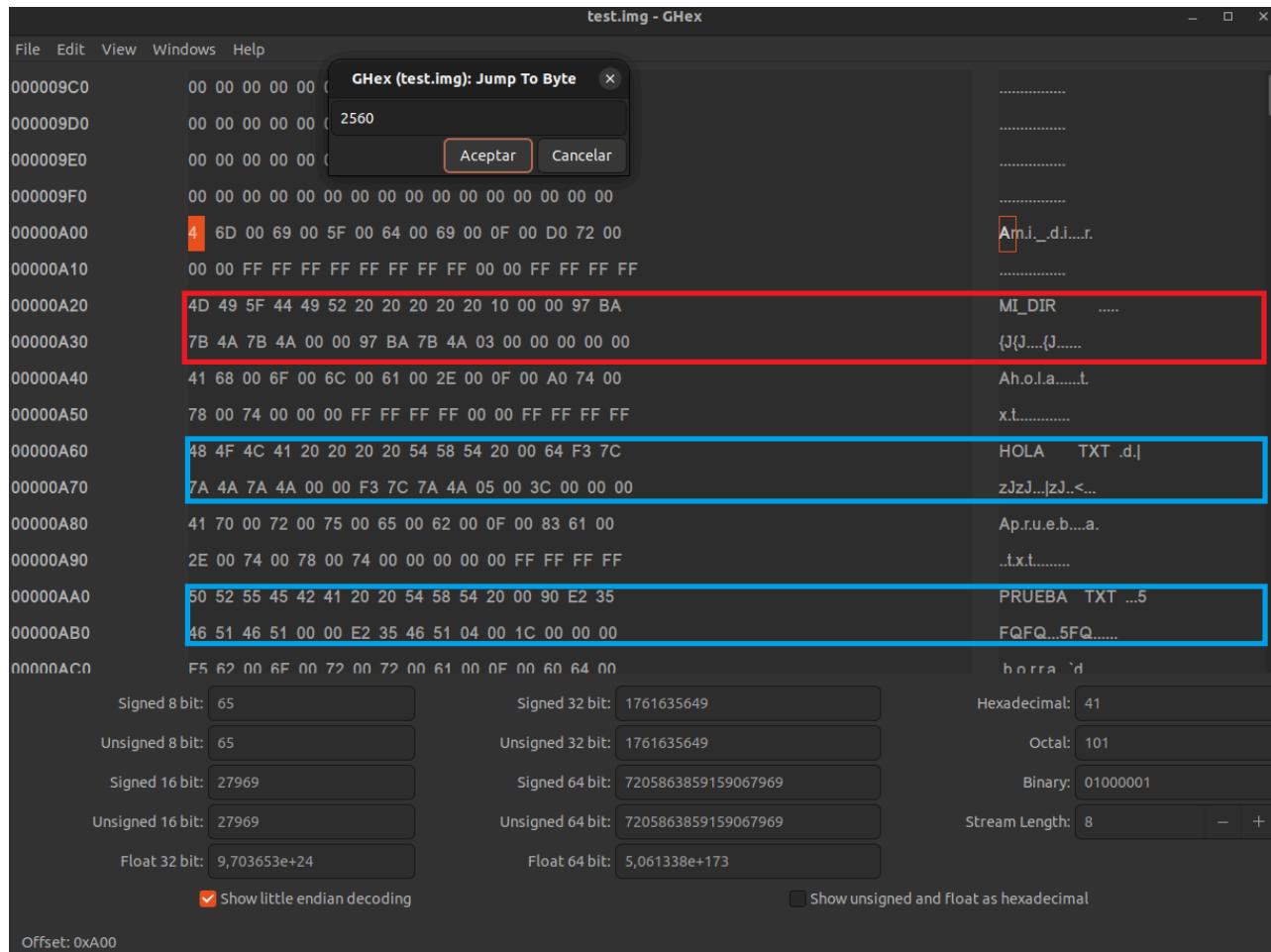
Para saber en qué byte comienza el contenido del root directory, recopilamos la información sobre las partes del FAT12 que están presentes antes de llegar al sector actual: primero tenemos el MBR que ocupa 512 bytes, y luego, contamos con el tamaño, en sectores, de las tablas FAT, que por cada una es de 2 sectores, y dado que cada sector ocupa 512 bytes, tenemos: $512B * 2 = 1024B$ de tamaño por tabla FAT. Al ser 2 tablas, concluimos en un tamaño de $1024B * 2 = 2048B$

De esta forma, podemos calcular el inicio, en byte, del root directory:

Byte de inicio del root directory = tamaño de MBR + tamaño de tablas FAT =

$$512B + 2048B = \mathbf{2560B}$$

De esta forma, en GHex, nos dirigimos al byte 2560 y vemos allí, al principio, información sobre los directorios y archivos que nos podemos encontrar en el root del filesystem.



Algunos datos sobre archivos y directorios del filesystem en el root directory

En la imagen, podemos visualizar, como primeros datos, que existe un directorio **mi_dir** (tenemos el dato de que es un directorio, si nos fijamos en el byte

11 de la entrada de 32 bytes, tiene valor ‘10’ que indica que es directorio) y dos archivos(tienen valor ‘20’ en el byte 11 de sus entradas): **hola.txt** y **prueba.txt**.

Para mostrar por programa los archivos del filesystem, debimos completar el archivo **read_root.c** para que encuentre y realice la correcta impresión de los archivos y directorios existentes y borrados.

```
marce998@marce-VM:~/Documentos/sor2/SoyR-2-Projects/FILESYSTEM$ ./read_root
Encontrada particion FAT12 0
En 0x200, sector size 512, FAT size 2 sectors, 2 FATs

Root dir_entries 512
[-]Directorio: mi_dir
----o----

[*]Archivo: vacio.txt \*\*\|


----o----
[*]Archivo: hola.txt

Hola, bienvenidos !
Pueden ahora tratar de leerme desde c !

[*]Archivo: prueba.txt

este es un archivo de prueba

[!]Archivo borrado: ?borrado.txt

Sin contenido, intente recuperar el archivo.

[!]Archivo borrado: ?.borraron.txt

Sin contenido, intente recuperar el archivo.

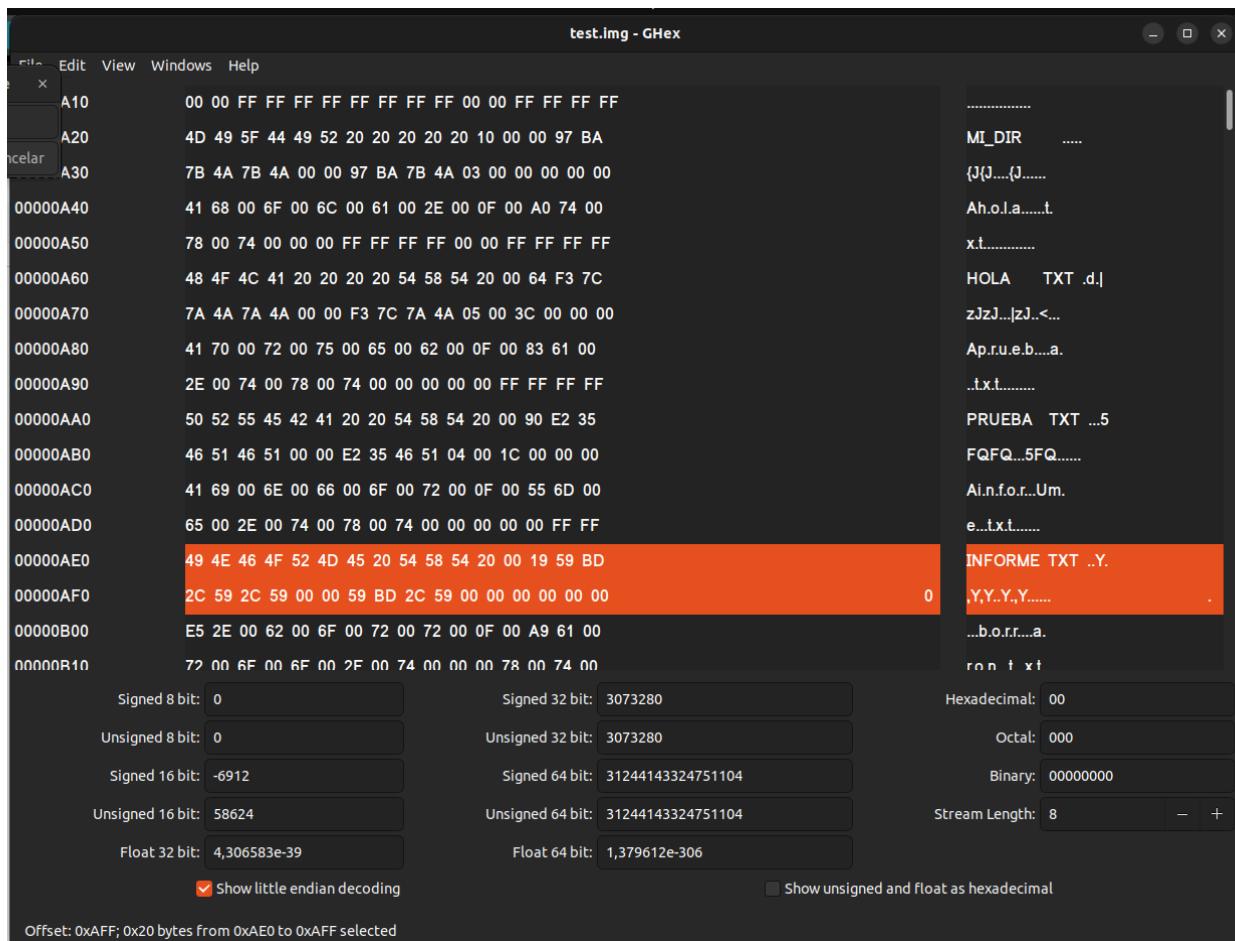
Leido Root directory, ahora en 0x4A00
marce998@marce-VM:~/Documentos/sor2/SoyR-2-Projects/FILESYSTEM$
```

*Ejecución del **read_root.c** para mostrar los archivos que hay en el filesystem*

Luego de esto, una de las tareas que debíamos realizar trataba de, una vez montado el filesystem, crear un archivo en el directorio raíz y borrarlo, para ver qué información sobre éste se encontraba tanto viendo desde GHex y por la ejecución del programa **read_root.c** completado antes. El archivo que creamos y borramos se llama **informe.txt**. A continuación mostraremos el proceso que realizamos para el borrado y visualización del archivo, tanto en GHex, como por ejecución del código.

```
marce998@marce-VM:~/Documentos/sor2/S0yR-2-Projects/FILESYSTEM$ ls -l /mnt
total 6
-rwxrwxrwx 1 root root 60 mar 26 2017 hola.txt
drwxrwxrwx 2 root root 2048 mar 27 2017 mi_dir
-rwxrwxrwx 1 root root 28 oct 6 2020 prueba.txt
marce998@marce-VM:~/Documentos/sor2/S0yR-2-Projects/FILESYSTEM$ cd /mnt
marce998@marce-VM: /mnt$ ls
hola.txt mi_dir prueba.txt
marce998@marce-VM: /mnt$ touch informe.txt
marce998@marce-VM: /mnt$ ls
hola.txt informe.txt mi_dir prueba.txt
marce998@marce-VM: /mnt$
```

*Creación del archivo **informe.txt** en la raíz del filesystem*

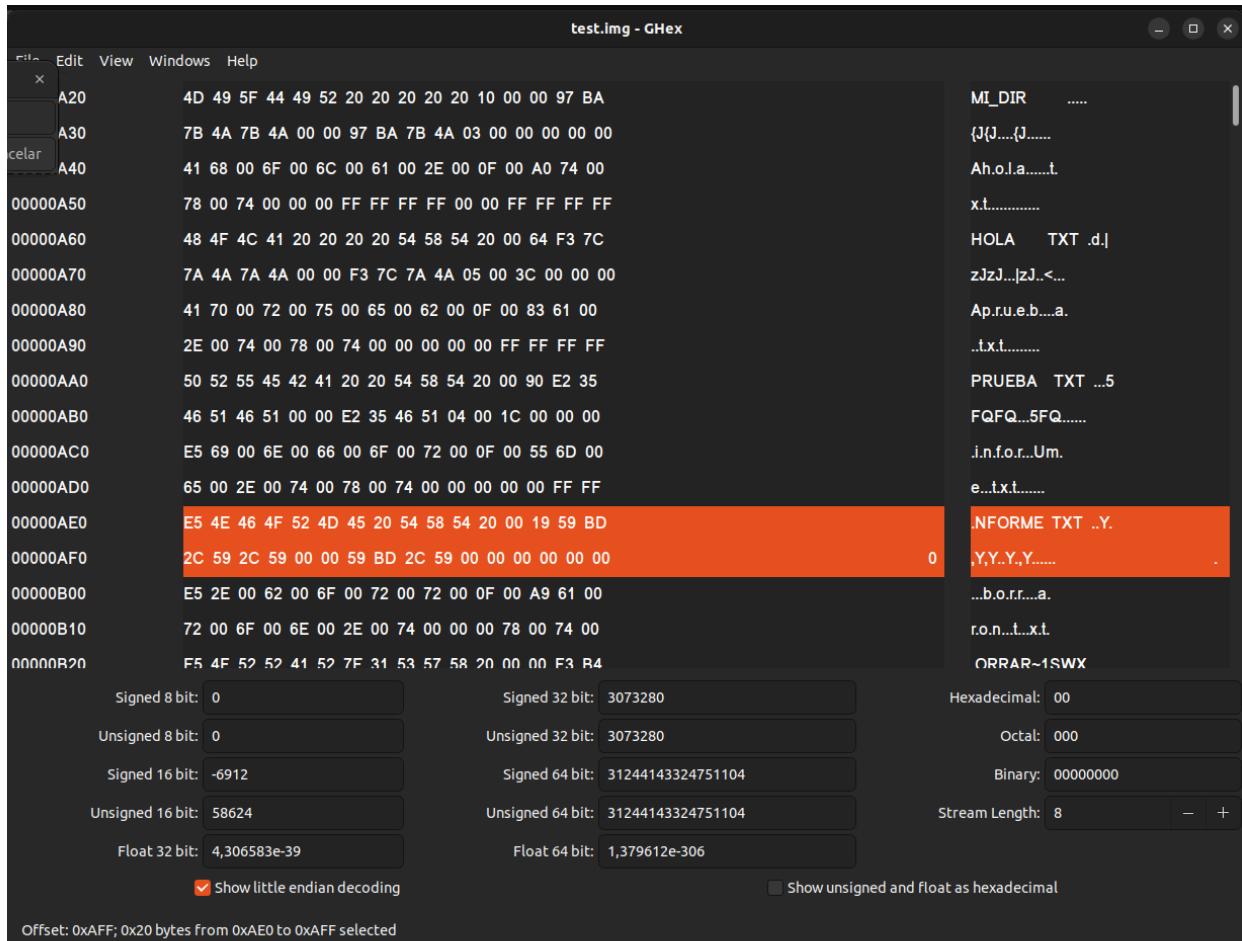


*Localización del archivo **informe.txt** (desde offset 0xAE0 al 0xAFF) en GHex*

```
marce998@marce-VM: /mnt$ rm informe.txt
marce998@marce-VM: /mnt$ ls
hola.txt mi_dir prueba.txt
marce998@marce-VM: /mnt$
```

*Eliminación del archivo **informe.txt***

Al eliminar el archivo, vemos en el editor que éste se mantiene en la misma ubicación (desde el byte 2784) pero cambió el valor de su primer byte a E5, marcando que es un archivo borrado.



Archivo **informe.txt** borrado (cambió su primer byte a E5)

A continuación, mostramos de nuevo la ejecución del programa de **read_root.c** para ver el archivo recientemente borrado.

```

narce998@narce-VM:~/Documentos/sor2/S0yR-2-Projects/FILESYSTEM$ ./read_root
Encontrada particion FAT12 0
En 0x200, sector size 512, FAT size 2 sectors, 2 FATs

Root dir_entries 512

[-]Directorio: mi_dir
----o----

[*]Archivo: vacio.txt "::::"

----o----
[*]Archivo: hola.txt

Hola, bienvenidos !
Pueden ahora tratar de leerme desde c !

[*]Archivo: prueba.txt

este es un archivo de prueba

[!]Archivo borrado: ?informe.txt
Sin contenido, intente recuperar el archivo.

[!]Archivo borrado: ?.borraron.txt
Sin contenido, intente recuperar el archivo.

Leido Root directory, ahora en 0x4A00
narce998@narce-VM:~/Documentos/sor2/S0yR-2-Projects/FILESYSTEM$ █

```

*Visualización del archivo borrado **informe.txt** ejecutando el compilado de **read_root.c***

Una vez realizado este proceso, se listan algunos detalles sobre el recupero de archivos que pudimos detectar:

- Si se sobreescribe el Directory Entry se pierde el nombre del archivo, atributos y fechas. Sin embargo, lo más importante que se pierde es la ubicación del primer cluster y tamaño de archivo. Complicando la tarea de recuperarlo ya que es difícil saber dónde comienza y que tan largo es.
- Si se sobreescribe algún LFN Directory Entry se pierde información de parte del nombre original del archivo por cada uno que se pierda.
- Si se sobreescriben los cluster con los datos de un archivo, se pierde el contenido.
- Al borrar un archivo de varios cluster, también se borran los valores correspondientes al archivo en la FAT. Por tanto el único cluster que se puede conocer del archivo es el primero, indicado en el Directory Entry.

Mientras no se sobreescriba el Directory Entry o el LFN Directory Entry que representen a un archivo, recuperar archivos que no ocupen más de un cluster se realiza de forma sencilla porque no es necesario buscar en la FAT. Caso contrario, si este archivo ocupase más de un cluster, se podría intentar buscar cluster que no tengan un archivo asignado en la FAT pero de todas formas no se conocería el orden correcto.

4. Leyendo Archivos

En esta parte del trabajo, en primer lugar, se nos pide montar el filesystem y crear dentro de la carpeta root/.. el archivo **lapapa.txt** e ingresarle un texto de relleno, para luego buscarlo en GHex y con el programa de la sección anterior(el que usamos es **read_root.c**).

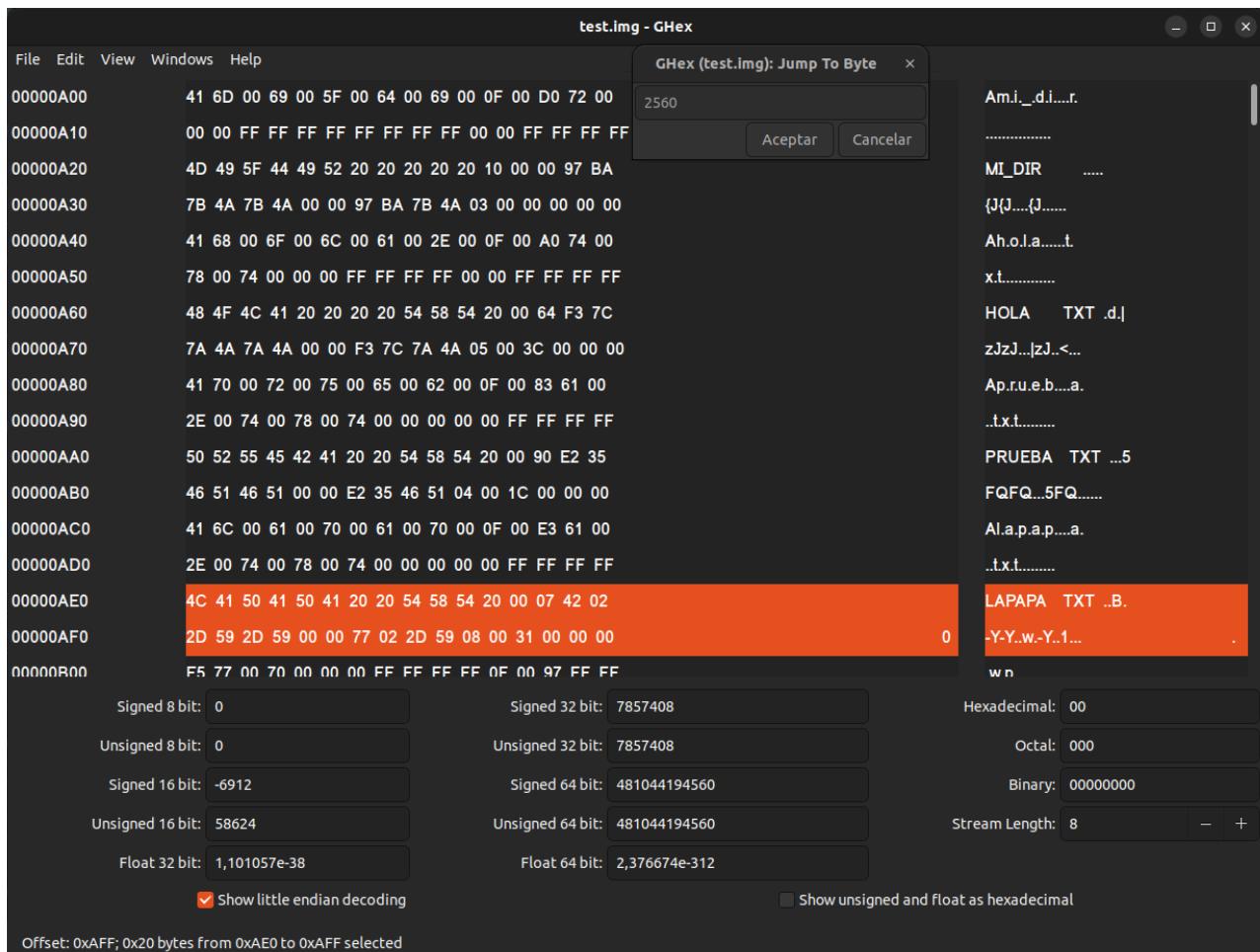
Montamos el filesystem con el comando : **sudo mount test.img /mnt -o loop,umask=000**.

Luego, nos dirigimos a la raíz del filesystem con **cd /mnt**, y allí creamos el archivo **lapapa.txt** con un texto de relleno.

```
marce998@marce-VM:/mnt$ ls
hola.txt  mi_dir  prueba.txt
marce998@marce-VM:/mnt$ touch lapapa.txt
marce998@marce-VM:/mnt$ ls
hola.txt  lapapa.txt  mi_dir  prueba.txt
marce998@marce-VM:/mnt$ nano lapapa.txt
marce998@marce-VM:/mnt$ cat lapapa.txt
Hola! este es un texto de relleno, buenas noches
marce998@marce-VM:/mnt$
```

*Entrando en el directorio root del filesystem y creando el archivo **lapapa.txt***

Una vez incorporado el archivo de texto, nos dirigimos al editor GHex para mostrar la ubicación del archivo y su contenido. Sabiendo que el root directory comienza en el byte 2560, y cuyas entradas son de 32 bytes, partimos desde ese byte, de a 32 ubicaciones, viendo cuál corresponde a nuestro archivo.



Entrada(32B) del archivo *lapapa.txt* en el root directory (desde el byte 2784 → offset 0xAE0)

Por su parte, para saber en qué byte comienza el contenido de *lapapa.txt*, debemos tener en cuenta dónde finaliza el root directory, el clúster indicado para el contenido del archivo y el tamaño por clúster. Sabiendo que en los bytes 26 y 27 de la entrada del archivo, en el root directory, nos da información sobre el clúster correspondiente a guardar los datos, vemos que sus valores son 08 y 00, es decir, en el clúster 8 es donde tenemos que buscar. En realidad, como los clústers 0 y 1 están reservados, debemos restarlos, con lo cual, la fórmula para calcular el byte de inicio del clúster, donde están los datos que buscamos, queda de la siguiente manera:

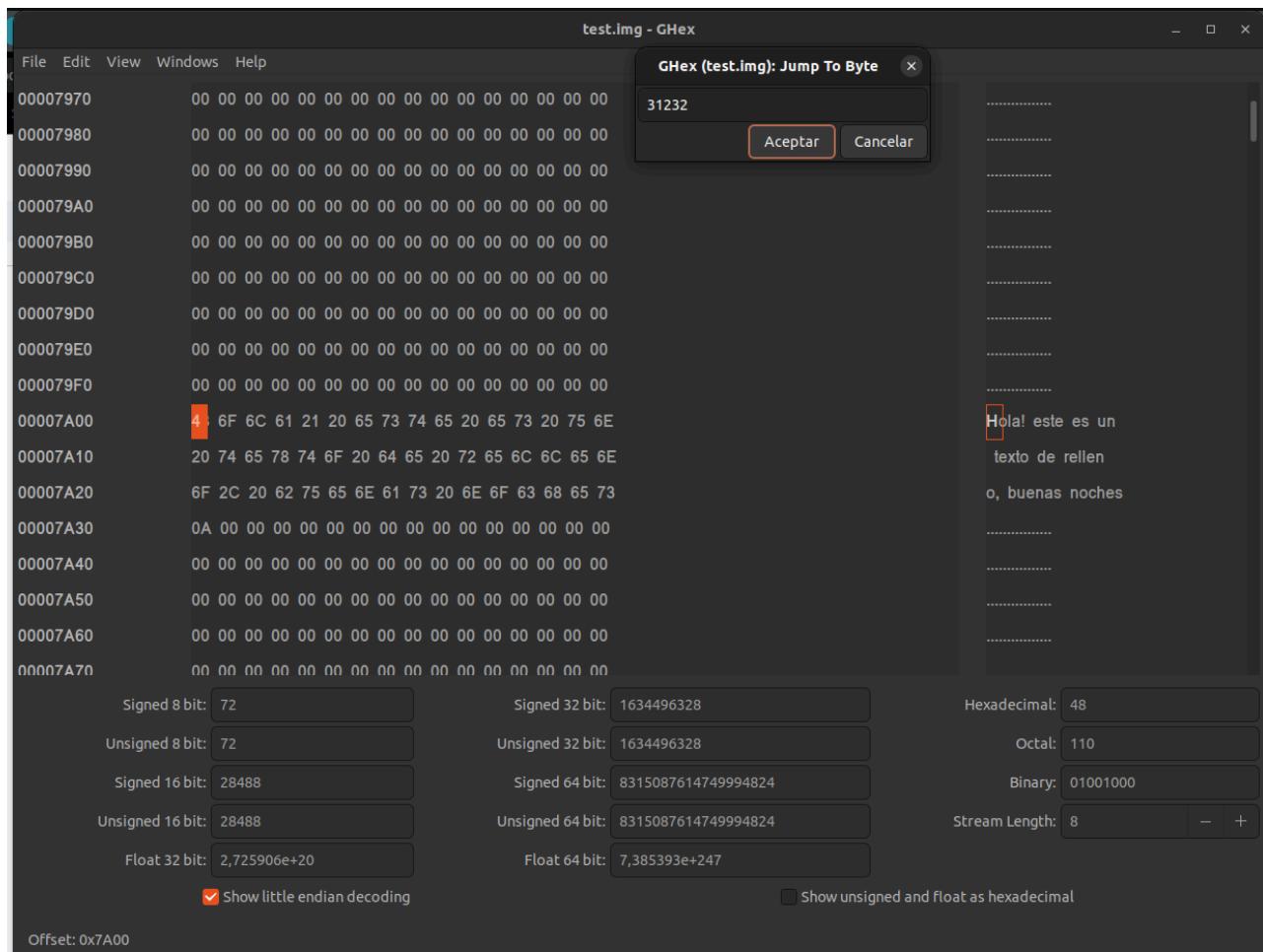
Byte de inicio del clúster = (número de clúster - 2) * tamaño de clúster + byte donde finaliza el root directory.

Para saber la finalización del root directory, basta con tener los datos de dónde comienza (byte 2560) más la cantidad de entradas que permite (dato del MBR: 512) por el tamaño de cada entrada (32 bytes): $2560B + (512 \times 32B) = 18944B$.

Y cada clúster tiene 4 sectores, cada uno de 512B, por lo que en total tiene 2048 bytes cada uno. Con esto, ya podemos terminar el cálculo:

$$\text{Byte de inicio del clúster} = (8-2)*2048\text{B} + 18944\text{B} = \mathbf{31232\text{B}}$$

A continuación, se deja evidencia tanto en el editor GHex y por el programa en c, del contenido de *lapapa.txt*



Byte donde comienza el contenido del archivo lapapa.txt

```
marce998@marce-VM:~/Documentos/sor2/S0yR-2-Projects/FILESYSTEM$ ./read_root
Encontrada particion FAT12 0
En 0x200, sector size 512, FAT size 2 sectors, 2 FATs

Root dir_entries 512

[-]Directorio: mi_dir
----o----

[*]Archivo: vacio.txt

----o----
[*]Archivo: hola.txt

Hola, bienvenidos !
Pueden ahora tratar de leerme desde c !

[*]Archivo: prueba.txt

este es un archivo de prueba

[*]Archivo: lapapa.txt

Hola! este es un texto de relleno, buenas noches
```

```
[!]Archivo borrado: ?.lapapa.txt.swp
sin contenido, intente recuperar el archivo.
```

```
Leido Root directory, ahora en 0x4A00
```

*Visualización del archivo **lapapa.txt** y su contenido ejecutando el compilado de **read_root.c***

A lo largo del trabajo se puede notar que, cuando leemos los archivos del filesystem se muestran archivos borrados. Por lo cual, para la última parte del trabajo se realizó un programa que permitiese recuperar un archivo borrado ingresando únicamente el nombre del archivo o parte de este.

Para esta prueba usaremos el archivo **lapapa.txt**, eliminandolo utilizando el comando “rm”². A continuación se muestra que al listar los archivos usando el programa **read_root** creado previamente, no se podrá ver el contenido de “lapapa.txt” ya que este se encuentra eliminado.

² “rm” es un comando de la familia de sistemas operativos Unix usada para eliminar archivos y directorios del sistema de archivos.

```
martin@DESKTOP-1DD67VP:~/Sor2-FAT$ ./read_root
Encontrada particion FAT12 0
En 0x200, sector size 512, FAT size 2 sectors, 2 FATS

Root dir_entries 512
[-]Directorio: mi_dir
----o----
[*]Archivo: vacio.txt

----o----
[*]Archivo: hola.txt

Hola, bienvenidos !
Pueden ahora tratar de leerme desde c !

[*]Archivo: prueba.txt
este es un archivo de prueba
[!]Archivo borrado: ?lapapa.txt
Sin contenido, intente recuperar el archivo.
[!]Archivo borrado: ?.lapapa.txt.swp
Sin contenido, intente recuperar el archivo.

Leido Root directory, ahora en 0x4A00
```

*archivo **lapapa.txt** eliminado*

La lógica de este nuevo programa se basa en ir recorriendo todos los archivos en el root directory (si el archivo es de tipo directorio, se recorren los archivos de este mismo). Luego, verifica si el primer byte es la marca de un archivo borrado, es decir 0xE5, para así trabajar únicamente con archivos que son borrados.

En cuanto al proceso de asociar el nombre del archivo con el texto que se solicita al usuario al principio del programa, se reutiliza la lógica de obtener los nombres de los programas creados previamente. Este consiste en aprovecharse que antes de cada Directory Entry hay uno o más LFN Directory Entries, y es de estos donde obtenemos el nombre (o una parte, si es un archivo de nombre largo). Por lo tanto, al llegar al Directory Entry ya cuenta con el nombre del archivo y este es el momento donde se realiza la comparación. Si el texto forma parte del nombre o es el nombre del archivo , y además, está eliminado, entonces se sobreescribe la marca de borrado con una de recuperado(en nuestro caso se decidió poner “A” para LFN y “R” para los otros). Finalmente, el programa muestra por pantalla los archivos

recuperados o un mensaje de que no se encontró ningún archivo con nombre ingresado.

En la imagen siguiente se muestra el resultado del programa llamado **recover_file** usando como texto ingresado “lapapa”, haciendo referencia al archivo que eliminamos recientemente para esta prueba.

```
martin@DESKTOP-1DD67VP:~/Sor2-FAT$ ./recover_file
Ingrese el nombre de archivo que deseé recuperar: lapapa
Encontrada particion FAT12 0
En 0x200, sector size 512, FAT size 2 sectors, 2 FATs

Root dir_entries 512

El archivo lapapa.txt ha sido recuperado, prueba listar los archivos del filesystem!
El archivo .lapapa.txt.swp ha sido recuperado, prueba listar los archivos del filesystem!
Leido Root directory, ahora en 0x4A00
```

*archivo **lapapa.txt** recuperado*

Como prueba final, listamos los archivos nuevamente con el programa **read_root** para ver los cambios.

```
martin@DESKTOP-1DD67VP:~/Sor2-FAT$ ./read_root
Encontrada particion FAT12 0
En 0x200, sector size 512, FAT size 2 sectors, 2 FATs

Root dir_entries 512

[-]Directorio: mi_dir
----o----

[*]Archivo: vacio.txt


----o----
[*]Archivo: hola.txt
Hola, bienvenidos !
Pueden ahora tratar de leerme desde c !

[*]Archivo: prueba.txt
este es un archivo de prueba
[*]Archivo: lapapa.txt
Hola! este es un texto de relleno, buenas noches
[*]Archivo: .lapapa.txt.swp
b0nano 6.2

Leido Root directory, ahora en 0x4A00
```

Destacamos que se recuperó un archivo .swp que también contiene “lapapa” en su nombre. Este es un archivo que crea el editor de texto **nano** al momento de

modificar el contenido en un archivo y sirve como copia de seguridad del archivo original. Su contenido muestra es un identificador (b0), el programa(nano) y por último la versión del programa.

Referencias

- Wikipedia. Design of the FAT File System.

https://en.wikipedia.org/wiki/Design_of_the_FAT_file_system

- C-JUMP. File System Data Structures.

<http://www.c-jump.com/CIS24/Slides/FileSysDataStructs/FileSysDataStructs.htm>

- Wikipedia. MBR - partition table entries

https://en.wikipedia.org/wiki/Master_boot_record#Partition_table_entries