SISTEMAS OPERATIVOS AVANZADOS

Práctica curso 2019-2020

Paula Rodríguez Delgado

Paula.rodriguez

ÍNDICE

Introducción

Esta práctica pide que se implemente un programa en c que permita leer la información de sistemas de ficheros Fat16 y Ext2.

Para la primera fase de esta práctica, se debe implementar la operación */info,* la cual, deberá recorrer el sistema de fichero, detectar el tipo que es, y según su tipo, mostrar información básica de dicho sistema. Solo se permitirán sistemas de ficheros que sean Ext2 o Fat16, cualquier otro tipo de volumen mostrará un mensaje de error.

La segunda fase de esta práctica consta en implementar la operación */find.* Para esta fase, hay que buscar un fichero dentro del volumen que nos indica. Si se encuentra dicho fichero, deberá mostrar el peso de dicho fichero en bytes; en caso contrario mostrará un mensaje de error.

La tercera fase de la práctica, muy similar a la anterior, también se tiene que implementar la operación */find,* solo que esta vez se buscará en cada sistema de ficheros en profundidad, es decir, no buscará solamente en la entrada de *root.*

La última fase de esta práctica consta en implementar la operación */delete.* El objetivo de esta fase es encontrar un fichero y eliminarlo, en cualquiera de los dos volúmenes.

Para realizar esta práctica no se nos ha saso ninguna restricción a nivel de uso de comandos i de funciones en C.

FAT 16

Estructura de datos

```
typedef struct {
    unsigned char* filesystem;
    unsigned char* systemName;
    unsigned short sizeSector;
    unsigned char clusterPerSectors;
    unsigned short reservedSectors;
    unsigned char numFat;
    unsigned short maxRootEntries;
    unsigned short fatSectors;
    unsigned char* label;

}fs_FAT;
```

Para guardar y mostrar la información de un volumen FAT16, se ha utilizado la estructura que aparece en la imagen. El motivo por el cual al final de todo se cambiaron los valores a *unsigned* es debido a que la documentación indica que, al no hacerlo *unsigned***,** algunos de los cálculos porían dar valores erróneos.

Fase 1 – Info

Lo primero que hay que hacer para detectar la información referente al sistema de ficheros Fat16, es comprobar realmente que este es un Fat16. Para determinarlo, nos tenemos que situar en el byte 54 del volumen. Si leemos los siguientes 8 bytes, por defecto la documentación indica que en este campo indica qué tipo de FAT es, ya sea FAT12, FAT16, FAT32

Una vez tenemos localizado que el volumen es tipo FAT16, pasamos a buscar los datos requeridos por el enunciado de la práctica.

```
lseek(fd, 3, SEEK_SET);                         //system name
fat.systemName = (char*)malloc(sizeof(char));
read(fd, fat.systemName, 8);
```

El nombre del volumen es un campo de texto de longitud 8 bytes

```
lseek (fd, 11, SEEK_SET);                       //sector size
read(fd, &fat.sizeSector, 2);
```

Campo que guarda el tamaño en bytes que ocupa cada sector.

```
lseek(fd, 13, SEEK_SET);                          //cluster
read(fd, &fat.clusterPerSectors, 1);
```

Campo que guarda el numero de *clusters* por sector.

```
lseek(fd, 14, SEEK_SET);                    //sectores reservados
read(fd, &fat.reservedSectors, 2);
```

Este campo para sistemas de ficheros Fat12 y Fat16 debe ser siempre 1. Guarda el número de sectores reservados.

```
lseek(fd, 16, SEEK_SET);                        //numero de fat's
read(fd, &fat.numFat, 1);
```

La documentación indica que no hay muchas tablas FAT en un volumen FAT

```
lseek(fd, 17, SEEK_SET);                    //max root entries
read(fd, &fat.maxRootEntries, 2);
```

Campo que guarda el número máximo de entradas que se pueden alojar en *root.*

```
lseek(fd, 22, SEEK_SET);                    //sectores por fat
read(fd, &fat.fatSectors, 2);
```

Guarda el tamaño de una tabla FAT. Para FAT 32 valdrá 0. Si se quiere detectar qué tipo de FAT es, se puede ver con este campo si es FAT32 u otro tipo de FAT

```
lseek(fd, 43, SEEK_SET);                     //volume label
fat.label = (char*)malloc(sizeof(char));
read(fd, fat.label, 11);
```

Guarda la etiqueta que se le ha puesto al volumen

EXT2

Estructura de datos

```
typedef struct {
    char* filesystem;
    InodesInfo inodeInfo;
    BlockInfo blockInfo;
    VolumeInfo volumeInfo;
}fs_ext2;
```

Para guardar y mostrar la información de un volumen EXT2, se ha utilizado la estructura que aparece en la imagen.

Fase 1 – Info

Antes de nada, explicar cómo saber si el volumen proporcionado es EXT2. Un sistema de ficheros Ext2, contendrá en la posición 56 sumándole la medida que tiene el *superblock* (1024), un campo de dos bytes, el cual, si contiene el valor 0xEF53, podemos afirmar que el volumen es de tipo Ext2.

Se ha dividido la estructura Ext2 en tres bloques, en los cuales encontraremos la información sobre los inodos, sobre los bloques, y sobre el propio volumen.

```
typedef struct {
    unsigned int inodeSize;
    unsigned int numInodes;
    unsigned int firstInode;
    unsigned int inodesGroup;
    unsigned int freeInodes;
}InodesInfo;
```

```
typedef struct {
    unsigned int blockSize;
    unsigned int numReservedBlocks;
    unsigned int numFreeBlocks;
    unsigned int numBlocks;
    unsigned int firstBlock;
    unsigned int blocksGroup;
    unsigned int fragsGroup;
}BlockInfo;
```

```
typedef struct {
    char* volumeName;
    time_t *lastCheck;
    time_t *lastMount;
    time_t *lastWrite;
}VolumeInfo;
```

Una vez tenemos localizado que el volumen es tipoEXT2, pasamos a buscar los datos requeridos por el enunciado de la práctica.

Información sobre los *inodos*

```
lseek(fd, 0 + superblock, SEEK_SET);          //num inodes
read(fd, &ext.inodeInfo.numInodes, 4);
```

Contiene el número de *inodes* que contiene el volumen.

```
lseek(fd, 16 + superblock, SEEK_SET);         //free inodes
read(fd, &ext.inodeInfo.freeInodes, 4);
```

Guarga el número de *inodes* libres del sistema de ficheros.

```
lseek(fd, 40 + superblock, SEEK_SET);         //inodes group
read(fd, &ext.inodeInfo.inodesGroup, 4);
```

Guarda el número de *inodes* que hay por grupo

```
lseek(fd, 84 + superblock, SEEK_SET);         //primer inode
read(fd, &ext.inodeInfo.firstInode, 4);
```

Guarda el índice al primer *inode* que se puede usar para guardar información

```
lseek(fd, 88 + superblock, SEEK_SET);          //inode size
read(fd, &ext.inodeInfo.inodeSize, 4);
```

Tamaño de un *inode.*

Información sobre los bloques

```
lseek(fd, 4 + superblock, SEEK_SET);          //numero de blocks
read(fd, &ext.blockInfo.numBlocks, 4);
```

Guarda el número de bloques totales en el volumen

```
lseek(fd, 8 + superblock, SEEK_SET);          //blocks reservados
read(fd, &ext.blockInfo.numReservedBlocks, 4);
```

El número de bloques reservados

```
lseek(fd, 12 + superblock, SEEK_SET);          //blocks libres
read(fd, &ext.blockInfo.numFreeBlocks, 4);
```

Número de bloques libres

```
lseek(fd, 20 + superblock, SEEK_SET);          //fist block
read(fd, &ext.blockInfo.firstBlock, 4);
```

Posición del primer bloque con información

```
lseek(fd, 24 + superblock, SEEK_SET);          //medida block
read(fd, &ext.blockInfo.blockSize, 4);
ext.blockInfo.blockSize = 1024 << ext.blockInfo.blockSize;
```

El tamaño en bytes de un bloque

```
lseek(fd, 32 + superblock, SEEK_SET);          //grupo block
read(fd, &ext.blockInfo.blocksGroup, 4);
```

Número de bloques que hay en cada grupo

```
lseek(fd, 36 + superblock, SEEK_SET);          //frags grou
read(fd, &ext.blockInfo.fragsGroup, 4);
```

El número de *fragments* que hay en cada grupo

Información sobre el volumen

```
ext.volumeInfo.lastMount = malloc(sizeof(time_t));
lseek(fd, 44 + superblock, SEEK_SET);          //last mount
read(fd, ext.volumeInfo.lastMount, 4);
```

Tiempo de la última vez que se montó el sistema de ficheros.

```
ext.volumeInfo.lastWrite = malloc(sizeof(time_t));
lseek(fd, 48 + superblock, SEEK_SET);          //last write
read(fd, ext.volumeInfo.lastWrite, 4);
```

Tiempo de la última vez que se escribió en el volumen

```
ext.volumeInfo.lastCheck = malloc(sizeof(time_t));
lseek(fd, 64 + superblock, SEEK_SET);          //last check
read(fd, ext.volumeInfo.lastCheck, 4);
```

La última vez que se realizó una comprobación del volumen

```
lseek(fd, 120 + superblock, SEEK_SET);          //volume name
read(fd, ext.volumeInfo.volumeName, 16);
```

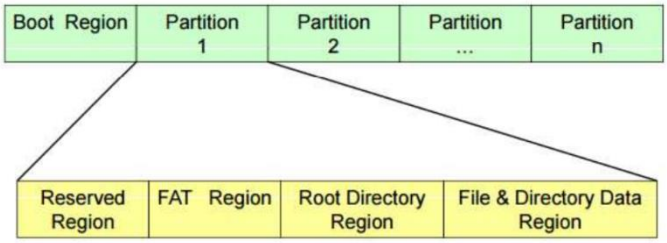
El nombre del volumen

Fase 2/3- Find

FAT

Para encontrar un fichero en un volumen FAT16, lo primero que hay que hacer es encontrar dónde se encuentra *root directory.*

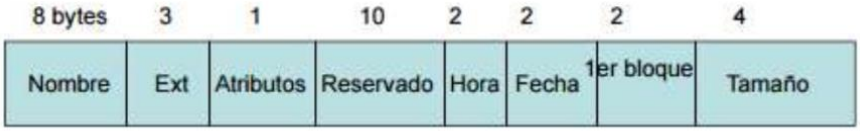
Para esto, tenemos en cuenta los datos encontrados en la fase1.



Tal y como vemos en la imagen, para llegar a la región de *root* debemos pasar primero por la región reservada y la región de FAT, entonces encontramos que el inicio de *root* se encuentra en:

```
rootDir = (fat.reservedSectors * fat.sizeSector) + (fat.numFat * fat.fatSectors * fat.sizeSector);
```

Las entradas que hay en *root* ocupan 32Bytes, los cuales se encuentran organizados de la siguiente manera:



Entonces, una vez se ha encontrado el inicio de *root directory,* y sabiendo que cada entrada tiene 32Bytes, tan solo hace falta iterar pasando por todas las *Root Entries*, revisando el nombre y la extensión, y el campo atributo que nos indica el tipo de entrada que es.

Archivo -> Atributos & 0x20

Directorio-> Atributos & 0x10

De esta forma, si encontramos una entrada que tenga el mismo nombre que se ha pasado por parámetro al ejecutar el programa, podemos afirmar que se trata de dicho fichero o directorio. Para saber el tamaño de dicho fichero, tenemos que acceder al último campo de la imagen anterior, la cual nos dice cuánto ocupa dicho fichero.

Para la fase 3, la búsqueda en profundidad, la mecánica para encontrar un fichero es la misma, sólo con una única diferencia. Una vez encontrado un directorio en *root directory*, para leer las entradas que contiene dicho directorio, hace falta posicionarnos sobre el directorio.

Para esto, basta con aplicar la siguiente fórmula:

```
entryposition = ((cluster - 2) * fat.clusterPerSectors) + fat.reservedSectors + (fat.numFat * fat.fatSectors);
entryposition = entryposition + ((fat.maxRootEntries * 32) + (fat.sizeSector - 1)) / fat.sizeSector;
entryposition = entryposition * fat.sizeSector;
```

Solo se accederá a una entrada que su *cluster* sea mayor a cero. Para encontrar la entrada, debemos restarle dos a este *cluster* y multiplicarlo por el número de *clusters* que contiene cada sector. A continuación, hay que sumarle todo el contenido que se encuentra entre la entrada de *root* y la entrada a este directorio.

Una vez se haya el valor de esta posición, la forma de iterar todos los archivos y la información que contiene dicho directorio, se realiza de la misma forma que se ha realizado con *root.*

EXT2

Para encontrar un fichero en un volumen EXT2, lo primero que hay que hacer es hallar dónde se encuentra la *Inodes Table.* Como indica la documentación de EXT2, el *Superblock* siempre comienza en la posición 1024 y este ocupa 1024 Bytes, independientemente del valor de *Block size.*

Como se puede ver en la imagen, para obtener la *Inodes Table* hay que moverse hacia la posición 1024 donde comienza el *Superblock*, y sumarle el tamaño de este, y obtenemos el *group descriptor.*

Según la documentación, si el *first block* es 1, el *group descriptor* se encontrará justo después del *superblock,* entonces hay que tenerlo en cuenta para poder encontrar la *inodes table.*

```
if (ext.blockInfo.firstBlock == 0) {
    lseek(fd, ext.blockInfo.blockSize + inodeTableAddr + 8, SEEK_SET);
    read(fd, &inodeTable, 4);
}
else {
    lseek(fd, SUPERBLOCK + SUPERBLOCK_SIZE + inodeTableAddr + 8, SEEK_SET);
    read(fd, &inodeTable, 4);
}
```

Donde el campo *inodeTableAddr* se encuentra de la siguiente forma:

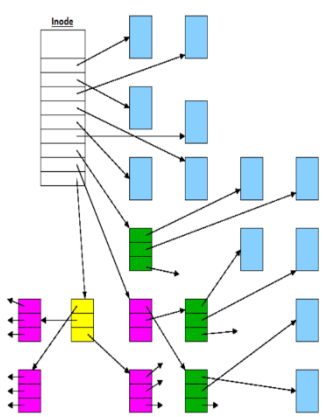
```
if (inode == 2) return 0;
else return ((inode - 1) / ext.inodeInfo.inodesGroup) * ext.blockInfo.blocksGroup * ext.blockInfo.blockSize;
```

Para buscar el la *root directory*, basta con pasar que el inodo valga dos, de esta forma el valor de *inodeTableAddr* valdrá 0.

Para encontrar dónde comienza el directorio de *root* o dónde comienza una entrada para la búsqueda recursiva, bastan con aplicar lo siguiente:

```
if (inode == 2) root = inodeTable * ext.blockInfo.blockSize + ext.inodeInfo.inodeSize;
else root = inodeTable * ext.blockInfo.blockSize + (ext.inodeInfo.inodeSize * position) + inodeTableAddr;
```

En el *Inode*, encontramos 12 punteros directos, 1 indirecto, 1 indirecto doble y un indirecto triple. Estos tres últimos no los leeremos. Para saber la posición de este bloque de datos, hay que multiplicar su número por el *Block size.* Esta organizado tal que así:



Comprobando bloque a bloque la información de los 12 punteros directos. Si uno de estos vale 0, significará que no apunta a ningún bloque, y que ya no hace falta que sigamos buscando porque el resto también estarán a 0.

En esta imagen se pueden ver los bloques de información (azules), el resto de colores se trata de los punteros indirectos.

Gracias al campo *rec_length* que tiene cada entrada, nos permite ir moviéndonos por cada entrada, con la acumulación de estos campos.

Cuando leemos el campo del nombre de cada entrada, si es igual al fichero del cual se requiere la búsqueda, el campo de tipo de entrada deberá ser 2, ya que si es uno es un directorio, y deberemos continuar con la búsqueda recursiva, y en el caso de que contenga otro valor, no nos interesará.

Para entrar dentro de un directorio, llamamos de forma recursiva a la función pasando como argumento el *inode* de dicho directorio. Para acceder a la información es de la misma forma que para *root.*

En el caso de llegar al final, y no encontrar dicho fichero, mostrará un mensaje de error.

Fase 4- delete

FAT

Para realizar la eliminación de un fichero que se encuentra en un volumen FAT, no ha sido especialmente difícil. Se ha reaprovechado la función de búsqueda de un fichero FAT, la cual nos devolverá la posición en la que se encuentra dicho fichero.

Si esta posición vale 0, significará que el fichero no ha sido encontrado. En caso contrario, se da pie a la eliminación.

Para eliminarlo es importante saber cuáles son los campos cruciales para encontrar el fichero. En la documentación, nos informaba que si el nombre de un fichero, en la primera casilla se encuentra el valor 0xE5, significa que el fichero ha sido eliminado. Ha sido el primer paso a realizar una vez se encuentra el fichero. Tras asignar el primer byte del nombre a este valor, se ha puesto el valor del *cluster* de esta entrada a 0, para que no se detecte información sobre dicho fichero.

EXT2

Para eliminar un fichero en EXT2 ha sido más complicado. No puedo afirmar con seguridad que se ha implementado de forma correcta, pero una vez eliminado el fichero ya no lo vuelve a encontrar, y se puede seguir buscando otros ficheros.

Para eliminar una entrada en *root*, lo que se hace primero es localizar dicha entrada a eliminar. Una vez encontrada, pasamos a sobrescribir dicha entrada, poniendo un 0 en cada campo innecesario.

Para las siguientes entradas que vienen a continuación, se va subiendo la posición de cada entrada, es decir, la siguiente entrada de la entrada eliminada, pasa a situarse en la posición de la eliminada, es decir, se va subiendo “una posición” cada entrada, hasta que llega al final de dicho bloque.

Pruebas realizadas

Fase 1

```
paula.rodriguez@montserrat:~/Shooter>shooter /info Ext2

 ------ Filesystem Information ------

Filesystem: EXT2
INFO INODE
Mida Inode: 128
NumInode: 2560
Primer Inode: 11
Inodes Grup: 1280
Inodes Lliures: 2542

INFO BLOC
Mida Bloc: 1024
Blocs Reservats: 512
Blocs Lliures: 9815
Total Blocs: 10240
Primer Bloc: 1
Blocs grup: 8192
Frags grup: 8192

INFO VOLUM
Nom Volum:
Ultima comprov: Mon Mar 23 13:55:34 2020
Ultim muntatge: Mon Mar 23 13:55:34 2020
Ultima escriptura: Mon Mar 23 13:55:34 2020
```

```
paula.rodriguez@montserrat:~/Shooter>shooter /info Fat16.bin

 ------ Filesystem Information ------

Filesystem: FAT16
System Name: mkdosfs
Mida del sector: 1024
Sectors Per Cluster: 1
Sectors reservats: 1
Numero de FATs: 2
MaxRootEntries: 512
Sectors per FAT: 16
Label: TEST2
```

```
paula.rodriguez@montserrat:~/Shooter>shooter fat16
Operacion incorrecta
```

Fase 2

```
paula.rodriguez@montserrat:~/Shooter>shooter /find Ext2 wep

Fitxer trobat. Ocupa 0 bytes.
```

```
paula.rodriguez@montserrat:~/Shooter>shooter /find Ext2 hh
Error. Fitxer no trobat.
```

```
paula.rodriguez@montserrat:~/Shooter>shooter /find Fat16.bin hh
Error. Fitxer no trobat.

paula.rodriguez@montserrat:~/Shooter>shooter /find Fat16.bin wait.h

Fitxer trobat. Ocupa 22 bytes.
```

```
paula.rodriguez@montserrat:~/Shooter>shooter /find Fat1 wait.h
Error. Volumen no encontrado
```

Fase 3

```
paula.rodriguez@montserrat:~/Shooter>shooter /find Ext2 pilotes

Fitxer trobat. Ocupa 0 bytes.
```

Al ser los nombres en FAT16 muy largos, no se ha buscado ningún fichero para probar su funcionalidad.

Fase 4

```
paula.rodriguez@montserrat:~/Shooter>shooter /delete Fat16.bin wait.h

El fitxer wait.h ha estat eliminat.

paula.rodriguez@montserrat:~/Shooter>shooter /find Fat16.bin wait.h
Error. Fitxer no trobat.
```

```
paula.rodriguez@montserrat:~/Shooter>shooter /delete Ext2 wep
El fitxer wep ha estat eliminat.

paula.rodriguez@montserrat:~/Shooter>shooter /find Ext2 wep
Error. Fitxer no trobat.
```

Conclusiones

A decir verdad, ha sido una práctica interesante de realizar. Muchas veces me he sentido frustrada realizando la práctica, ya que por un despiste, la práctica deja de funcionar por completo.

La sección a la que más tiempo se le ha dedicado ha podido ser, la búsqueda recursiva en un volumen EXT2, ya que no se tenia en cuenta que hay que sumar también todo lo que hay previamente a la entrada, y no se realizaba esta búsqueda de forma correcta.

Se a aprendido a ver la estructura de dos tipos de volúmenes y ha sido interesante ver otras formas diferentes a las que está uno acostumbrado de ver de cómo se guardan los datos, y a saber también como poder editarlos.