PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE INGENIERÍA
INF239 SISTEMAS OPERATIVOS

LABORATORIO 5

TEMA: SISTEMA DE ARCHIVOS FAT

Objetivos

El objetivo básico del laboratorio es comprender y reconocer las estructuras que forman parte del sistema de archivos FAT (12/16/32) accediendo directamente a ellas mediante programas en Lenguaje C.

Durante el laboratorio al alumno se le presentará una imagen de un sistema de archivo FAT y deberá escribir un programa que resuelva la situación propuesta.

Descripción

El usuario promedio tiene un conocimiento limitado acerca de cómo realmente se administra los datos guardados en su computadora. Es cierto que no es necesario que esté enterado, le basta con saber que sus datos están guardados en archivos y estos a su vez en carpetas, y tanto unos como otros pueden ser copiados, renombrados o borrados. Sin embargo para llevar a cabo estas operaciones administrativas se necesita de cierta “infraestructura”, entrecomillo esta palabra por que al final de cuentas lo que está en el disco son bytes y nada más que bytes, sin embargo si se establece por consenso algún estructura, entonces podemos darle una interpretación a esta secuencia de bytes. Los sistemas de archivos que han perdurado tradicionalmente son FAT12, FAT16, FAT32, ext2, ext3, ext4, MINIX 3, NTFS, ISO9660, etc. Los sistemas operativos pueden soportar uno o más sistemas de archivos, esto depende de cómo están diseñados. Cada sistema de archivo tiene una estructura específica y las llamadas al sistema son escritas de acuerdo al sistema de archivo. Como el título de este laboratorio lo indica el sistema de archivo que nos ocupa es FAT.

Qué es una imagen de disco?

Este concepto es importante y será lo primero que explicaremos. Todo sistema de archivo contenido en un disco, CD, pendrive, etc. se le puede dividir en dos partes: la parte administrativa, algunos autores le denominan metadata (data de la data) y la parte de los datos propiamente dichos.

Cuando se guarda un archivo (imagen, hoja de cálculo, texto, etc) en un dispositivo de almacenamiento, este dispositivo debe contener un sistema de archivos. El archivo va al área de datos, pero el sistema de archivo necesita saber en qué lugar específico lo guardó, además de otros atributos que le permitan manipularlo. Los programas que manejan este archivo no saben nada acerca de el lugar exacto en que se guardó ni de los atributos, esto es tarea del sistema operativo y se lo piden como un servicio a través de las llamadas al sistema para manejo de archivos.

Una vez que el archivo es copiado, al contenido se le ha asignado espacio en la data del sistema de archivos y sus atributos en la parte de la metadata. Por supuesto, dentro de los atributos se encuentra el lugar donde se encuentra el contenido.

Cuando se realiza una copia hacia otro dispositivo, el sistema operativo debe de conocer también el sistema de archivo destino. Un caso común es que ambos contengan el mismo sistema de archivos, pero esto no es obligatorio. Debido a que el dispositivo de destino difícilmente tendrá el mismo tamaño, es poco probable que el archivo se copie en el mismo sitio. Si los sistemas de archivos son distintos, el sistema operativo se encargará de hacer las equivalencias correspondientes de los atributos en el sistema de archivos destino.

Entonces, qué es la imagen de un disco. Es el contenido del sistema de archivos completo. Es decir, si se copiara byte por byte, a otro dispositivo del mismo tamaño, obtendríamos una copia exacta del dispositivo origen. Es otras palabras los datos y los metadatos se encontrarán en el mismo sitio en que se encontraban en el dispositivo origen. Una cosa diferente hubiera sido copiar los archivos uno por uno o por bloque. Los archivos y sus metadatos estarían ubicados en sitios completamente distintos. Por ese motivo algunos acostumbran a comparar la imagen de un sistema de archivos como una foto del mismo.

En Linux copiar un archivo (o un bloque de ellos) lo puede llevar a cabo con la orden *cp* desde una terminal o usando el *mouse* desde el explorador de archivos (*Nemo* en caso de Linux Mint). Sin embargo copiar un sistema de archivos no se puede hacer con *cp*, en este caso puede hacer uso de la orden *dd*.

En esta oportunidad no estamos interesados en crear imágenes de sistema de archivos. Junto con este material se le proporciona algunas imágenes y el objetivo es analizar las estructuras de la FAT contenidas en estas imágenes. Todas las imágenes contienen un sistema de archivo FAT

Imágenes proporcionadas junto con este archivo:

disqueteFAT12.img.zip
usbFAT16.img.zip
usbFAT32.img.zip

Descomprima los archivos antes de usarlos.

FAT conceptos y estructura de datos

El espacio contenido en el dispositivo siempre está dividido en sectores (512 bytes). Es decir la capacidad útil del dispositivo está medida en sectores. Así por ejemplo el disquete de 3’ 1/4 tenía capacidad de 2880 sectores.

El área correspondiente a la data, tienen como unidad lógica a los *clusters*. Un *cluster* está formado por una cantidad de sectores (el número de sectores queda determinado en el momento que se le da formato al dispositivo) . El número de sectores por *cluster* se encuentra en el *boot sector*.

Los *clusters* inician su numeración con el número 2. El sector donde se encuentra el *cluster* 2 (primer *cluster* de datos) se encuentra en el *boot sector*.

El sistema de archivos FAT es uno de los más sencillos y está formado por:

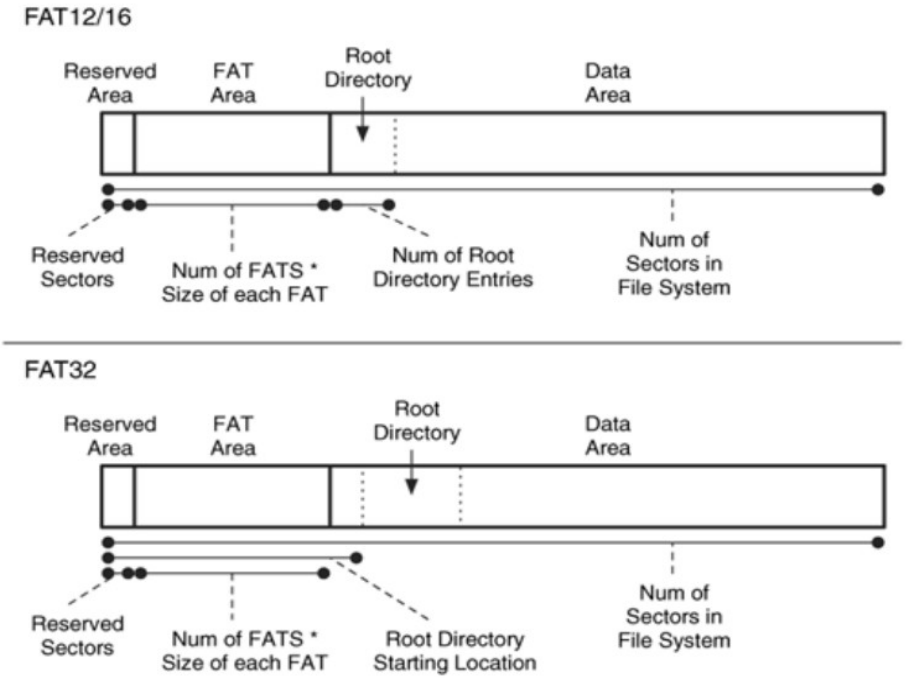
a) El *boot sector*, está ubicado en el primer sector del dispositivo. En él se encuentra información acerca del sistemas de archivo, tales como, el número de sectores reservados, el número de sectores de la FAT, el número de FAT's, ubicación del directorio raíz (FAT12 y FAT16), el número del sector donde inicia la data, etc.

b) La FAT, es la secuencia de *clusters* que indican dónde se encuentra el contenido por cada archivo

c) El directorio raíz, son entradas que contiene información de un archivo y el primer *cluster* del archivo.

d) El área de los datos, formado por los *clusters*.

Esquema del sistema de archivo FAT



Tomado del libro *File System Foresinc Analysis by Carrier*

El directorio raíz en FAT12 y FAT16 formaban parte de la metadata, es decir su tamaño era fijo y siempre se encontraba a continuación de la segunda FAT. Por tanto el *cluster* 2 se encontraba después del directorio raíz. Los subdirectorios contenidos en el directorio raíz si se encontraban en el área de datos. En cambio en FAT32 el tamaño del directorio raíz podía crecer porque estaba ubicada en el área de datos y su ubicación no era a continuación de la segunda FAT, sino que estaba a un número de sectores del inicio del dispositivo. Este número se encontraba en el *boot sector*. Además el directorio raíz se encontraba en el *cluster* 2.

Tomado del libro *File System Foresinc Analysis by Carrier*

Leyendo el boot sector

En la literatura al boot sector también se le conoce como BIOS Parameter Block (BPB). Los 27 primeros bytes es igual a todas las FAT's. A partir de ahí empieza una extensión de la BPB, pero es diferente en 16 y FAT32. A continuación se presenta la estructura tomada de la página:

https://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html

A continuación la estructura del boot sector de la FAT12 que también es común a la FAT16 y FAT32

```
Bytes   Content
0-2     Jump to bootstrap (E.g. eb 3c 90; on i86: JMP 003E NOP.
        One finds either eb xx 90, or e9 xx xx.
        The position of the bootstrap varies.)
3-10    OEM name/version (E.g. "IBM  3.3", "IBM 20.0", "MSDOS5.0", "MSWIN4.0".
        Various format utilities leave their own name, like "CH-FOR18".
        Sometimes just garbage. Microsoft recommends "MSWIN4.1".)
        /* BIOS Parameter Block starts here */
11-12   Number of bytes per sector (512)
        Must be one of 512, 1024, 2048, 4096.
13      Number of sectors per cluster (1)
        Must be one of 1, 2, 4, 8, 16, 32, 64, 128.
        A cluster should have at most 32768 bytes. In rare cases 65536 is OK.
14-15   Number of reserved sectors (1)
        FAT12 and FAT16 use 1. FAT32 uses 32.
16      Number of FAT copies (2)
17-18   Number of root directory entries (224)
        0 for FAT32. 512 is recommended for FAT16.
19-20   Total number of sectors in the filesystem (2880)
        (in case the partition is not FAT32 and smaller than 32 MB)
21      Media descriptor type (f0: 1.4 MB floppy, f8: hard disk; see below)
22-23   Number of sectors per FAT (9)
        0 for FAT32.
24-25   Number of sectors per track (12)
26-27   Number of heads (2, for a double-sided diskette)
28-29   Number of hidden sectors (0)
        Hidden sectors are sectors preceding the partition.
        /* BIOS Parameter Block ends here */
30-509  Bootstrap
510-511 Signature 55 aa
```

A continuación la estructura del boot sector de la FAT16

```
11-27   (as before)
28-31   Number of hidden sectors (0)
32-35   Total number of sectors in the filesystem
        (in case the total was not given in bytes 19-20)
36      Logical Drive Number (for use with INT 13, e.g. 0 or 0x80)
37      Reserved (Earlier: Current Head, the track containing the Boot Record)
        Used by Windows NT: bit 0: need disk check; bit 1: need surface scan
38      Extended signature (0x29)
        Indicates that the three following fields are present.
        Windows NT recognizes either 0x28 or 0x29.
39-42   Serial number of partition
43-53   Volume label or "NO NAME    "
54-61   Filesystem type (E.g. "FAT12   ", "FAT16   ", "FAT     ", or all zero.)
62-509  Bootstrap
510-511 Signature 55 aa
```

Por último la estructura del boot sector de la FAT32

```
11-27   (as before)
28-31   Number of hidden sectors (0)
32-35   Total number of sectors in the filesystem
36-39   Sectors per FAT
40-41   Mirror flags
        Bits 0-3: number of active FAT (if bit 7 is 1)
        Bits 4-6: reserved
```

```
        Bit 7: one: single active FAT; zero: all FATs are updated at runtime
        Bits 8-15: reserved
42-43   Filesystem version
44-47   First cluster of root directory (usually 2)
48-49   Filesystem information sector number in FAT32 reserved area (usually 1)
50-51   Backup boot sector location or 0 or 0xffff if none (usually 6)
52-63   Reserved
64      Logical Drive Number (for use with INT 13, e.g. 0 or 0x80)
65      Reserved - used to be Current Head (used by Windows NT)
66      Extended signature (0x29)
        Indicates that the three following fields are present.
67-70   Serial number of partition
71-81   Volume label
82-89   Filesystem type ("FAT32   ")
```

A continuación un programa que lee el *boot sector* de un sistema de archivo con FAT12.

```c
/**********   structfat.h   ************/

typedef struct {
    unsigned char jmp[3];
    char oem[8];
    unsigned short sector_size;
    unsigned char sectors_per_cluster;
    unsigned short reserved_sectors;
    unsigned char number_of_fats;
    unsigned short root_dir_entries;
    unsigned short total_sectors_short; // if zero, later field is used
    unsigned char media_descriptor;
    unsigned short fat_size_sectors;
    unsigned short sectors_per_track;
    unsigned short number_of_heads;
    unsigned short hidden_sectors;
    char boot_code[480];
    unsigned short boot_sector_signature;
} __attribute((packed)) Fat12BootSector;
```

```c
/*************   leebootFAT12.c   **********/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "structfat.h"

int main(int narg, char *argv[])
{   Fat12BootSector boot;
    int fd;

    if(narg != 2) {
        printf("Usage; %s <name image disc>\n",argv[0]);
        exit(1);
    }
    if((fd=open(argv[1], O_RDONLY)) < 0)
         perror("No pudo abrir imagen de disco");
    if(read(fd,&boot, sizeof(boot))< 0)
         perror("No pudo leer imagen de disco");
    printf("Jump code: %x:%x:%x\n",boot.jmp[0],boot.jmp[1],boot.jmp[2]);
    printf("OEM code: %s\n",boot.oem);
    printf("sector_size: %d\n", boot.sector_size);
    printf("sectors_per_cluster: %d\n", boot.sectors_per_cluster);
    printf("reserved_sectors: %d\n",boot.reserved_sectors);
    printf("number_of_fats: %d\n",boot.number_of_fats);
    printf("root_dir_entries: %d\n",boot.root_dir_entries);
    printf("total_sector_short: %d\n",boot.total_sectors_short);
    printf("media_descriptor: %d\n", boot.media_descriptor);
    printf("fat_size_sectors: %d\n", boot.fat_size_sectors);
    printf("sectors_per_track: %d\n", boot.sectors_per_track);
    printf("number_of_heads: %d\n",boot.number_of_heads);
    printf("hidden_sectors: %d\n",boot.hidden_sectors);
    printf("boot_sector_signature: %x\n",boot.boot_sector_signature);
    close(fd);
    return 0;
}
```

Su compilación y su ejecución.

```
alejandro@abdebian:2020-2$ gcc -o leebootFAT12 leebootFAT12.c
alejandro@abdebian:2020-2$ ./leebootFAT12 disqueteFAT12.img
Jump code: eb:3e:90
OEM code: 2'}IcIHC
sector_size: 512
sectors_per_cluster: 1
reserved_sectors: 1
number_of_fats: 2
root_dir_entries: 224
total_sector_short: 2880
media_descriptor: 240
fat_size_sectors: 9
sectors_per_track: 18
number_of_heads: 2
hidden_sectors: 0
boot_sector_signature: aa55
alejandro@abdebian:2020-2$
```

Tarea

1) Escriba un programa que lea el *boot sector* de un sistema de archivos con FAT16, donde el nombre de la imagen del disco se pasa como argumento por la línea de ordenes.
2) Escriba un programa que lea el *boot sector* de un sistema de archivos con FAT32, donde el nombre de la imagen del disco se pasa como argumento por la línea de ordenes.
3) Escriba un programa que lea el *boot sector* de un sistema de archivos con FAT, donde el nombre de la imagen del disco se pasa como argumento por la línea de ordenes. El programa debe determinar si la imagen contiene un sistema de archivos FAT12, FAT16 o FAT32.

El directorio raíz

La estructura de un directorio es la siguiente:

```
Bytes   Content
0-10    File name (8 bytes) with extension (3 bytes)
11      Attribute - a bitvector. Bit 0: read only. Bit 1: hidden.
        Bit 2: system file. Bit 3: volume label. Bit 4: subdirectory.
        Bit 5: archive. Bits 6-7: unused.
12-21   Reserved (see below)
22-23   Time (5/6/5 bits, for hour/minutes/doubleseconds)
24-25   Date (7/4/5 bits, for year-since-1980/month/day)
26-27   Starting cluster (0 for an empty file)
28-31   Filesize in bytes
```

Añadimos al archivo *structfat.h* la estructura de la entrada de directorio.

```
21 typedef struct {
22     unsigned char filename[8];
23     unsigned char ext[3];
24     unsigned char attributes;
25     unsigned char reserved[10];
26     unsigned short modify_time;
27     unsigned short modify_date;
28     unsigned short starting_cluster;
29     unsigned int file_size;
30 } __attribute((packed)) FatEntry;
31
32
```

Además por los datos obtenidos del programa anterior y la teoría antes explicada en un sistema con FAT12 necesitamos recorrer 19 sectores (1 del boot, 9 de la primera FAT y 9 de la segunda FAT) para acceder al directorio raíz. También se conoce que la raíz contiene exactamente 224 entradas.

En la entrada de directorio, los 8 bytes desde el offset 0x00 hasta 0x07 representan el nombre del archivo. El primer byte del nombre indica su estado. Usualmente, este contiene un carácter normal, pero hay valores especiales:

```
0x00
      Filename never used.
0xe5
      The filename has been used, but the file has been deleted.
0x05
      The first character of the filename is actually 0xe5.
0x2e
      The entry is for a directory, not a normal file. If the second byte is also 0x2e, the cluster
      field contains the cluster number of this directory's parent directory. If the parent directory
      is the root directory (which is statically allocated and doesn't have a cluster number),
      cluster number 0x0000 is specified here.
Any other character
      This is the first character of a real filename.
```

Si el nombre de archivo contiene menos de ocho caracteres, es completado con espacios en blanco.

La extensión comprende los siguientes tres caracteres. No hay caracteres especiales. Note que el punto no se almacena. Si la extensión es de menos de tres caracteres, el resto se llena con espacios en blanco.

Con esta información podemos construir el siguiente programa, que lee las entradas del directorio raíz de un sistemas de archivos de FAT12.

```c
/************   leeDirRaiz.c   **********/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "structfat.h"

#define     SIZE_BLOCK   512
#define     OFFSET        19

int main(int narg, char *argv[])
{   FatEntry entry;
    int i,fd;

    if(narg != 2) {
        printf("Usage; %s <name image disc>\n",argv[0]);
        exit(1);
    }
    if((fd=open(argv[1], O_RDONLY)) < 0)
         perror("No pudo abrir imagen de disco");
    lseek(fd,OFFSET*SIZE_BLOCK,SEEK_SET);
    for(i=0; i< 224; i++){
        if(read(fd,&entry, sizeof(entry))< 0)
             perror("No pudo leer imagen de disco");
        if((entry.attributes & 0x8) && (entry.attributes & 0x20)){
            printf("Volume label: [%.8s]\n",entry.filename);
            continue;
        }
        /* No aseguramos que no se una entrada de nombre largo y que sea un archivo */
        if((entry.attributes != 0x0F) && (entry.attributes & 0x20))
            printf("File [%.8s.%.3s]  First cluster [%d] Size [%d]\n",\
                 entry.filename,entry.ext,entry.starting_cluster,entry.file_size);
    }
    close(fd);
    return 0;
}
```

Compilando y ejecutando:

```
alejandro@abdebian:2020-2$ gcc -o leeDirRaiz leeDirRaiz.c
alejandro@abdebian:2020-2$ ./leeDirRaiz disqueteFAT12.img
Volume label: [INF239  ]
File [COLLAT~1.ZIP]  First cluster [5] Size [434]
File [COUNTP~1.ZIP]  First cluster [6] Size [379]
File [CREDO   .PDF]  First cluster [7] Size [36261]
File [LECTURE1.PDF]  First cluster [78] Size [22900]
File [PROCPY  .PDF]  First cluster [123] Size [76562]
File [SCO6E   .PDF]  First cluster [273] Size [677074]
alejandro@abdebian:2020-2$
```

TAREA

1) En el programa *leeDirRaiz.c* el tamaño de la FAT se le asignó directamente 9 sectores. Sin embargo, esto no funciona para FAT16. Lo más apropiado es leer el tamaño de la FAT en el *boot sector*. Modifique su programa para que lea el directorio raíz de un sistema de archivos FAT12 como FAT16, leyendo el tamaño de la FAT desde el *boot sector*.

2) Como se explicó anteriormente, en FAT12 y FAT16, el directorio raíz es de tamaño fijo (no puede crecer) el tamaño se encuentra en el *boot sector* y además se ubica después de la segunda FAT. Para el caso de la FAT32 el directorio raíz ya no se encuentra inmediatamente después de la segunda FAT. Sino que el desplazamiento está especificado en el *boot sector*. Escriba un programa para que lea el directorio raíz de un sistema de archivos FAT32.

VFAT

En Windows 95 se introdujo una variación: VFAT (Virtual FAT), esta es la FAT junto con nombre de archivos largos (LFN), que pueden llegar hasta 255 bytes de longitud.

3) Investigue en internet la estructura de la entradas de archivos de nombres largos. Elabore un programa en C que lea los nombres cortos y los nombres largos del directorio raíz de cualquier sistema de archivo. Note que la VFAT es debido al sistema operativo más no al sistema de archivo. Es decir pueden haber sistemas de archivos FAT12 con nombres largos.

Entradas de la FAT

La cadena de los *clusters* que forman el contenido del archivo se encuentra en la FAT. Usted puede pensar la FAT como un arreglo con entradas. Si es FAT12, las entradas son de 12 bits (1.5 bytes). Si es FAT16 las entradas son de 16 bits (2 bytes) y si es FAT32 las entradas son de 32 bits (4 bytes). En cada entrada se guarda el número del siguiente *cluster*. Como se dijo anteriormente, el primer *cluster* de un archivo se encuentra en la entrada del directorio correspondiente, los siguientes *clusters* se sacan de la FAT.

En cualquier tipo de FAT la entrada 0 y la entrada 1 no se usan para almacenar *clusters*. Recién la entrada 2 corresponde al cluster 2.

La secuencia termina con un -1.

El siguiente programa muestra la cadena de *clusters* del primer archivo del directorio raíz, para el sistema de archivo proporcionado (*usbFAT16.img*).

```c
/*************   leeClusters.c   **********/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "structfat.h"

#define     SIZE_BLOCK   512
#define     OFFSET       513

int main(int narg, char *argv[])
{   FatEntry entry;
    int i,fd,offset;
    short cluster;

    if(narg != 2) {
        printf("Usage; %s <name image disc>\n",argv[0]);
        exit(1);
    }
    if((fd=open(argv[1], O_RDONLY)) < 0)
         perror("No pudo abrir imagen de disco");
    lseek(fd,OFFSET*SIZE_BLOCK,SEEK_SET);
    for(i=0; i< 512; i++){
        if(read(fd,&entry, sizeof(entry))< 0)
             perror("No pudo leer imagen de disco");
        if((entry.attributes & 0x8) && (entry.attributes & 0x20)){
            printf("Volume label: [%.8s]\n",entry.filename);
            continue;
        }
        /* No aseguramos que no se una entrada de nombre largo y que sea un archivo */
        if((entry.attributes != 0x0F) && (entry.attributes & 0x20)) {
            printf("File [%.8s.%.3s]  First cluster [%d] Size [%d]\n",\
                 entry.filename,entry.ext,entry.starting_cluster,entry.file_size);
            cluster = entry.starting_cluster;
            printf("Cadena de clusters: %d ",cluster);
            while(1) {
                offset = 512 + cluster*2;
                lseek(fd,offset,SEEK_SET);
                read(fd,&cluster,sizeof(cluster));
                if(cluster==-1) break;
                printf("%d ",cluster);
            }
            printf("\n");
            break;
        }
    }
    close(fd);
    return 0;
}
```

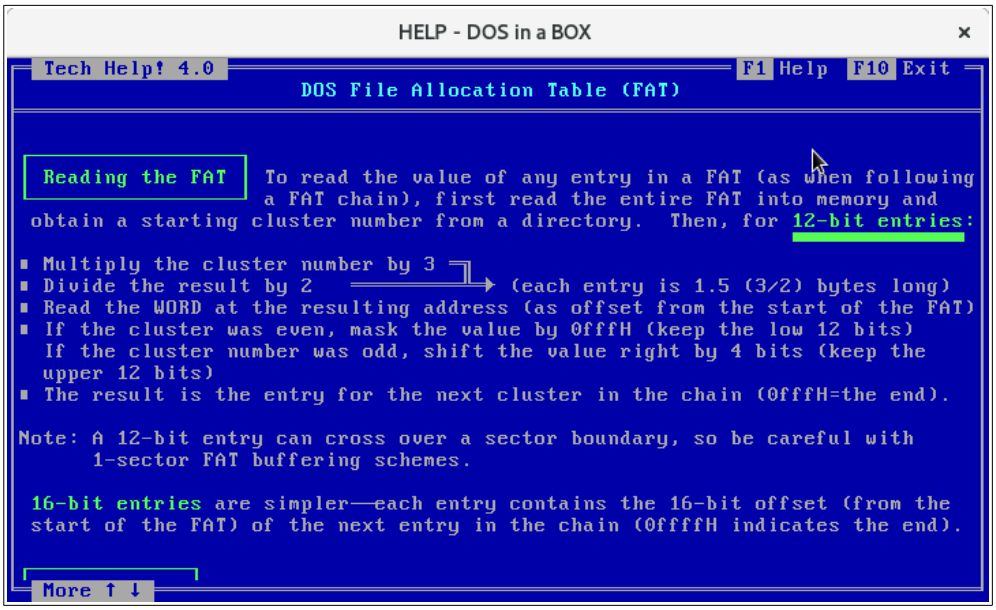
Al compilarlo y ejecutarlo, se obtiene:

```
alejandro@abdebian:2020-2$ gcc -o leeClusters leeClusters.c
alejandro@abdebian:2020-2$ ./leeClusters usbFAT16.img
File [CARRIE~1.PDF]  First cluster [3] Size [388063]
Cadena de clusters: 3 4 5 6 7 8
alejandro@abdebian:2020-2$
```

TAREA

1) Modifique el programa para que muestre las cadenas de *clusters* de todos los archivos contenidos en la raíz. Por ahora solo muestra la cadena del primer archivo.
2) Modifique el programa para que el offset sea obtenido a partir del *boot sector*. De esta forma el programa podrá ser usado para distintos sistemas de archivos de FAT16.
3) Para el caso de FAT32, la entrada ya no se multiplica por 2 sino por 4. Elabore un programa para que determine el tipo de FAT (16 o 32) y muestre la cadena de *clusters* de todos los archivos del directorio raíz.

En caso de la FAT 12, determinar la cadena de *clusters* de la FAT, es más engorroso. En principio no hay tipo de dato de 1.5 bytes. Así que se tendrá que leer dos bytes (short) y eliminar 4 bits con una mascara. La siguientes indicaciones le pueden ayudar a leer la FAT12



4) Escriba un programa que lea el directorio raíz de un sistema de archivos con FAT12 e imprima las cadenas de *clusters* de cada archivo que hay en el directorio raíz.

Cómo crear distintos sistemas de archivos con FAT desde Linux Mint y luego modificarlo

El siguiente procedimiento permitirá obtener la imagen de un sistema de archivo FAT16 y a continuación montarlo sobre */media/$USER/<ID>*

Se necesita tener instalado el paquete *udisks2*, en las computadoras de los laboratorios ya se encuentra instalado.

A continuación preparamos un archivo con sistema de archivo FAT16, desde una terminal escriba:

```
dd if=/dev/zero of=./FAT16D1.img bs=1024 count=32768

mkfs.vfat -F 16 FAT16D1.img
```

Si desea obtener un sistema de archivo con FAT32 solo cambie 16 por 32.

Ahora procedemos a montarlo, debe conocer la ruta absoluta del archivo imagen. Asumamos que se encuentra en */home/alulab/Documentos*, entonces

```
udisksctl loop-setup -f /home/alulab/Documentos/FAT16D1.img
```

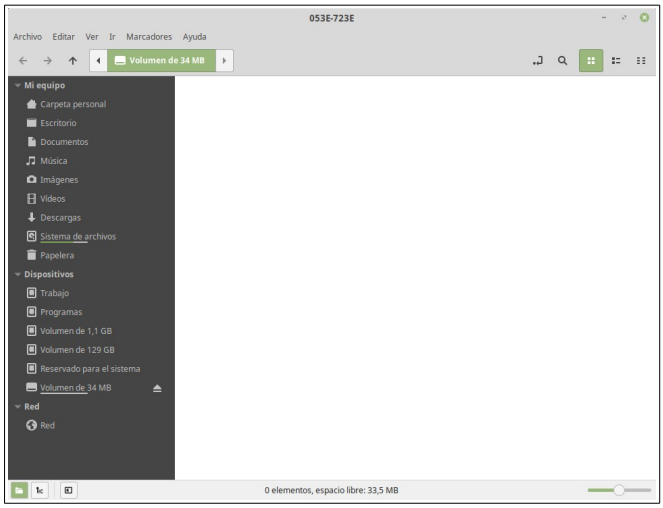
Como respuesta deberá aparecer en la terminal los siguiente:

```
Mapped file /home/alulab/Documentos/FAT16D1.img as /dev/loop0.
```

El sistema crea un directorio con nombre igual a algo como 053E-723E (el nombre puede variar) en */media/alulab/* (asumiendo que el usuario es *alulab*) y monta sobre este directorio la imagen del disco. El directorio creado tiene como propietario al que ejecutó el comando (*alulab* en este caso), de forma que puede copiar y borrar archivos en esta imagen. Puede ver su contenido de forma acostumbrada:

```
ls -l /media/alulab/053E-723E
```

También se abrirá (en Linux Mint 20) una ventana del navegador de archivos (Nemo) mostrando el directorio donde se ha montado la imagen.



En adelante cuando ingrese a este directorio, estará accediendo a este disco virtual. Cuando ya no se desee acceder más a esta imagen se debe desmontarlo para que los cambios tomen efecto.

```
udisksctl unmount -b /dev/loop0
```

La idea de enseñarle este procedimiento, es que usted pueda crear y modificar su propia imagen, muy aparte de las imágenes que se le proporciona. De esta forma podrá experimentar con sistemas de archivos FAT de distinto tamaño.

Prof. Alejandro T. Bello Ruiz