Sistemas Operativos y Redes 2

Trabajo Práctico 1

Estudiante

Martin Bornholdt - mbornholdt3@gmail.com
Lucas Hector Esteban Garcia Espinosa - lucasge.1994@gmail.com

Profesores

Agustin Alexander
Pedro Gutierrez

Descripción

En este informe, se detalla el análisis de una imagen de un sistema de archivos FAT12, proveída junto con la consigna del trabajo práctico. Siguiendo esta consigna, hemos estudiado el contenido del sistema de archivos a bajo nivel, usando el editor hexadecimal Bless, y generamos código en C para mostrar el contenido por consola y para modificarlo según lo pedido.

Instrucciones

Junto a este documento se entregan los archivos con el código fuente pedido en cada punto. Estos archivos pueden compilarse con el GNU Compiler Collection (GCC).

gcc (nombre del fuente) -o (nombre del ejecutable)

Esto genera un archivo compilado ejecutable con el nombre ingresado. Luego, para ejecutarlos utiliza el nombre que utilizó para referenciarlo.

./(nombre del ejecutable)

Para el punto 4C, debe crear un archivo, eliminarlo, y luego, correr el objeto compilado, pasando como parámetro el nombre del archivo.

./(nombre del ejecutable) (nombre del archivo eliminado)

Respuestas

Se ha puesto umask=000 para que el directorio donde se monta la imagen, al igual que los subdirectorios y archivos que se creen dentro del mismo, tengan permisos de lectura, escritura y ejecución para todos los usuarios.

La orden y función umask (abreviatura de user mask) funciona en entornos POSIX, establece los permisos por defecto para los nuevos archivos y directorios creados por el proceso actual. El valor en hexadecimal 000 representa a los permisos de lectura, escritura y ejecución

Cargando el MBR

Bytes	Descripción
0-445	Código de booteo
446-461	Entrada nro. 1 de la tabla de particiones
462-477	Entrada nro. 2 de la tabla de particiones
478-493	Entrada nro. 3 de la tabla de particiones
494-509	Entrada nro. 4 de la tabla de particiones
510-511	Fin del MBR (0xAA55)

Tabla 1: Estructura del MBR.

En la tabla 1 podemos ver la estructura del MBR. Al principio del MBR, encontramos el código de booteo. Este código contiene información sobre el disco, como el nombre del fabricante en ASCII, el número de tablas FAT, y otros valores listados en la tabla 2. En la figura 1 se muestra los valores de los primeros 60 bytes del código de booteo del filesystem, y en la figura 2 podemos ver estos valores como caracteres ASCII.

Bytes	Propósito
0-2	Instrucciones de salto a código de booteo
3-10	Nombre del fabricante en ASCII
11-12	Bytes por sector
13	Sectores por Cluster
14-15	Tamaño de área reservada, en sectores
16	Cantidad de tablas FAT
17-18	Número máximo de archivos en root
19-20	Sectores en filesystem. Si es mayor a 2B se setea a 0 y se guarda en 32-35
21	Tipo de dispositivo (0xF0 = disco extraible, 0xF8 disco fijo)
22-23	Tamaño de FAT, en sectores
24-25	Sectores por pista en disco físico
26-27	Número de cabezas en disco físico
28-31	Número de sectores antes del inicio de la partición
32-35	Sectores en filesystem. 0 si los bytes 19-20 no son iguales a 0
36	Número de dispositivo para interrupción 13h
37	Sin uso
38	Firma extendida para validar los siguientes 3 campos (0x29)
39-42	Serial del volumen
43-53	Etiqueta del volumen, en ASCII
54-61	Tipo de filesystem en ASCII, generalmente FAT o FAT12
62-509	Sin uso
510-511	Fin del MBR (0xAA55)

Tabla 2: Estructura del boot sector en FAT12

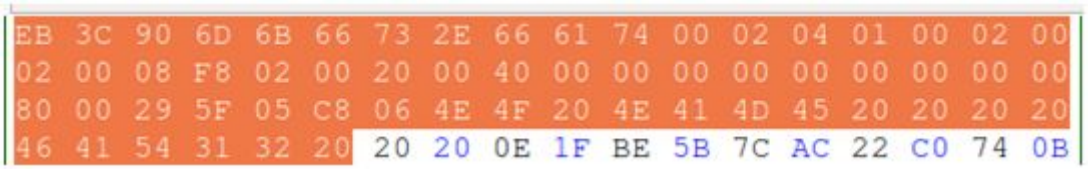

Fig. 1: Primeros bytes del MBR, en hexadecimal

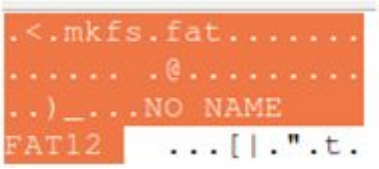

Fig. 2: Primeros bytes del MBR, en ASCII

En la tabla 1 podemos ver la ubicación de la tabla de particiones. La estructura de las entradas de la tabla de particiones se detalla en la tabla 3. La figura nro. 3 muestra una captura de la primera entrada de la tabla de particiones del filesystem en hexadecimal. Podemos ver que el primer byte, correspondiente al flag de booteable, está seteado en 0x80. Esto significa que la partición es booteable.

Bytes	Descripción
0-0	Flag de booteable
1-3	Dirección inicial de CHS
4-4	Tipo de partición
5-7	Dirección final de CHS
8-11	Dirección de LBA inicial
12-15	Tamaño en sectores

Tabla 3: Estructura de una entrada de la tabla de particiones.

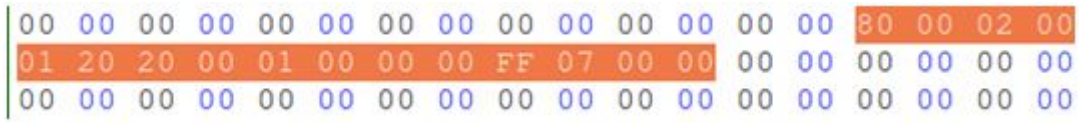

Fig. 3: Primera entrada de la tabla de particiones, en hexadecimal.

En la figura 4 podemos ver la ubicación correspondiente a las otras 3 entradas de la tabla de particiones. Estas entradas están en blanco, por lo que podemos afirmar que hay una sola partición.

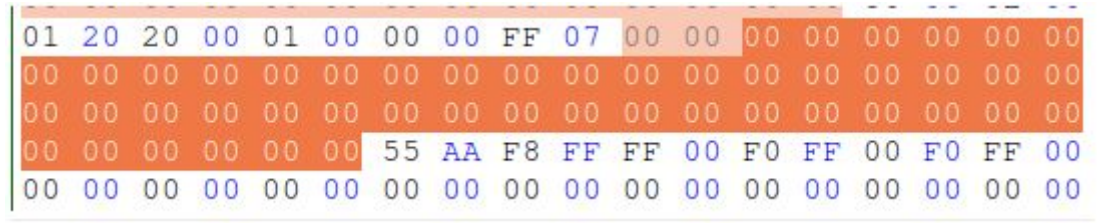

Fig. 4: Entradas 2, 3 y 4 de la tabla de particiones, en hexadecimal.

En el código, tenemos definidas estructuras con los tamaños de algunos elementos del filesystem, como PartitionTable y Fat12BootSector. Al momento de recorrer el archivo por código, obtenemos el byte inicial como referencia, y a partir de eso podemos ir saltando de una posición a otra en el archivo según los tamaños de las estructuras. En la figura 5 podemos ver el output del archivo ejecutable que muestra las entradas de la tabla de particiones por consola.

```
lucas@lucas-AM1M-S2H:~/Descargas/TP-Garcia$ ./2d
Partition entry 0: First byte 80
  Partition start in CHS: 00:02:00
  Partition type 0x01
  Partition end in CHS: 00:20:20
  Relative LBA address 0x00000001, 2047 sectors long
Partition entry 1: First byte 00
  Partition start in CHS: 00:00:00
  Partition type 0x00
  Partition end in CHS: 00:00:00
  Relative LBA address 0x00000000, 0 sectors long
Partition entry 2: First byte 00
  Partition start in CHS: 00:00:00
  Partition type 0x00
  Partition end in CHS: 00:00:00
  Relative LBA address 0x00000000, 0 sectors long
Partition entry 3: First byte 00
  Partition start in CHS: 00:00:00
  Partition type 0x00
  Partition end in CHS: 00:00:00
  Relative LBA address 0x00000000, 0 sectors long
```

Fig. 5: Tabla de particiones, en consola.

```
lucas@lucas-AM1M-S2H:~/Descargas/TP-Garcia$ ./2d
Partition entry 0: First byte 80
  Partition start in CHS: 00:02:00
  Partition type 0x01
  Partition end in CHS: 00:20:20
  Relative LBA address 0x00000001, 2047 sectors long
```

Fig. 5-a: Primera partición, en consola.

Cargando la tabla de archivos

Bytes	Propósito
0	Primer carácter del nombre (0x00 = sin asignar, 0xE5 = archivo borrado)
1-10	Caracteres 2-11 del nombre del archivo
11-11	Atributos del archivo(solo lectura, oculto, etc.)
12-12	Reservada
13-13	Tiempo en que se creó el archivo (décimas de segundo)
14-15	Hora de creación (horas, minutos, segundos)
16-17	Fecha de creación
18-19	Fecha de último acceso
20-21	2 bytes más significativos de la dirección del primer cluster (0x0 para FAT12)
22-23	Tiempo de modificación (horas, minutos, segundos)
24-25	Fecha de modificación
26-27	2 bytes menos significativos de la dirección del primer cluster
28-31	Tamaño del archivo (0 para directorios)

Tabla 4: Estructura de un archivo en root

```
alumno@alumno-virtualbox:~/Descargas/entregable$ gcc read_boot.c -o read_boot
alumno@alumno-virtualbox:~/Descargas/entregable$ ./read_boot
Partiion type: 1
Encontrado FAT12 0
  Jump code: EB:3C:90
  OEM code: [mkfs.fat]
  sector_size: 512
  sectors_per_cluster: 4
  size_reserved_area: 1
  number_of_fats: 2
  max_number_files: 512
  sectors_in_fs: 2048
  media_type: 248
  size_of_fat: 2
  sectors_per_track: 32
  number_of_heads: 64
  sectors_before_start_partition: 0x00000000
  sectors_in_filesystem: 0x00000000
  int_13h_drive: 128
  extended_boot_signature: 41

volume_id: 0x06C8055F
  Volume label: [NO NAME    ]
  Filesystem type: [FAT12   ]
  Boot sector signature: 0xAA55
```

Fig. 6: mensaje por consola de read_boot.c.

Para leer los archivos en el sistema, podemos ver el archivo con el editor hexadecimal, empezando desde el principio del sector Root (desde el principio del archivo, más el tamaño de los sectores anteriores y el tamaño de las tablas FAT multiplicado por la cantidad de tablas FAT). En la figura 7 podemos ver el archivo HOLA.TXT en el editor hexadecimal, y en la figura 8 podemos ver el mismo archivo en formato ASCII.

```
00 6C 00 61 00 2E 00 0F 00 A0 74 00 78 00 74 00 00 00
FF FF FF FF 00 00 FF FF FF FF 48 4F 4C 41 20 20 20 20
54 58 54 20 00 64 F3 7C 7A 4A 7A 4A 00 00 F3 7C 7A 4A
05 00 3C 00 00 00 E5 2E 00 73 00 77 00 70 00 00 00 0F
00 A1 FF FF FF FF FF FF FF FF FF FF FF FF 00 00 FF FF
```

Fig. 7: archivo hola.txt en root, en hexadecimal.

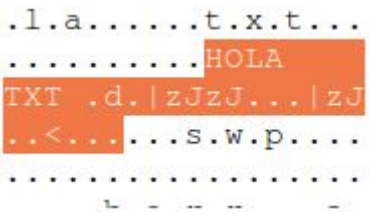

Fig. 8: Archivo hola.txt en root, en ASCII.

Para verlo por código, hacemos lo mismo: empezando desde cero, le sumamos el tamaño del boot sector y el de las tablas FAT. Una vez ahí, leemos siguiendo la estructura detallada en la tabla 4, tantas veces como entradas tenga root (este dato se encuentra en el boot sector). En la figura 9 podemos ver el output del código que lee los archivos en el file system.

```
lucas@lucas-AM1M-S2H:~/Escritorio/Reentrega$ ./3a
FAT12 filesystem found from partition 0
En  0x200, sector size 512, FAT size 2 sectors, 2 FATs

Root dir_entries 512
File: [Am.]
Subdirectory: [MI_DIR .    ]
Directory: [.       .   ]
Directory: [..      .   ]
File: [Av.]
File: [VACIO   .TXT]
File: [Ah.]
File: [HOLA    .TXT]
Deleted file: [?..]
Deleted file: [?..]
Deleted file: [?ORRAR~1.SWP]
Deleted file: [?..]
Deleted file: [?..]
Deleted file: [?ORRAR~1.SWX]

Root directory read, now at 0x4A00
lucas@lucas-AM1M-S2H:~/Escritorio/Reentrega$
```

Fig. 9: lista de archivos, por código y en consola.

Fig. 10: Lista de archivos, en hexadecimal y en ASCII.

Probando ahora con nuevo archivo de nombre paraBorrar.txt. Después de crear este archivo en root lo borraremos. Buscándolo con la función de búsqueda de Bless podemos verlo en hexa como se muestra en la figura 11, y en ASCII como se muestra en la figura 12.

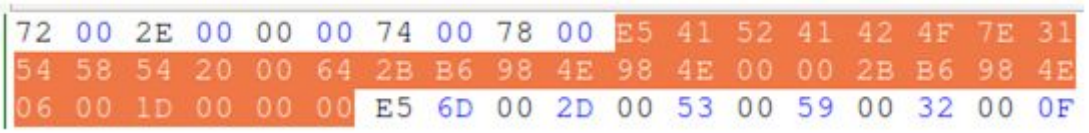

Fig. 11: Archivo "paraBorrar.txt", en hexadecimal.

Fig. 12: Archivo "paraBorrar.txt", en ASCII.

Si ejecutamos el código del punto anterior podremos ver el archivo borrado como se muestra en la figura 13, aunque con el nombre trunco.

```
lucas@lucas-AM1M-S2H:~/Documentos/SOR/2020 - 1/entregable$ ./3a
FAT12 filesystem found from partition 0
En  0x200, sector size 512, FAT size 2 sectors, 2 FATs

Root dir_entries 512
File: [Am.]
Subdirectory: [MI_DIR .    ]
File: [Ah.]
File: [HOLA    .TXT]
File: [Ad.]
Subdirectory: [DIR1    .    ]
File: [BORRAR~.SWP]
File: [B..]
File: [B..]
File: [BORRAR~.SWX]
Deleted file: [?t.◆◆◆]
Deleted file: [?p.]
Deleted file: [?ARABO~1.TXT]
Deleted file: [?m.]
Deleted file: [?..]
Deleted file: [?OUTPU~1.    ]

Root directory read, now at 0x4A00
```

Fig. 13: Archivo "paraBorrar.txt", ahora borrado. Borrado y recuperación de archivos.

En FAT12, para borrar un archivo, se reemplaza el primer carácter por el valor hexadecimal 0xE5, que indica que este archivo fue borrado. Si quisiéramos recuperar el archivo, tendríamos que reemplazar el primer carácter por otro. Si el valor del número de cluster que se encuentra en el archivo y la entrada en la tabla FAT no fueron sobrescritos, entonces el archivo se puede recuperar.

Lectura de Archivos

Para leer los archivos, debemos ver el número de cluster inicial y las entradas en la tabla FAT. Si el archivo solo tiene una entrada en la tabla, la totalidad del archivo está contenido en el primer cluster. Una vez que encontramos un archivo, debemos ver el tamaño en bytes. Para probar esto, creamos un archivo de nombre "lapapa.txt". Podemos ver este archivo en hexa en la figura 14 y un listado generado por el código del punto 3a en la figura 15.

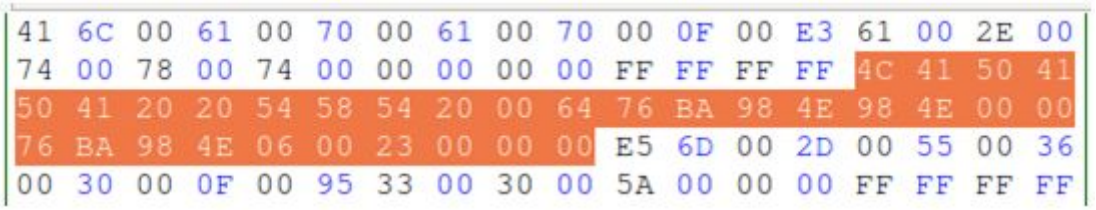

Fig. 14: Archivo "lapapa.txt", en hexadecimal.

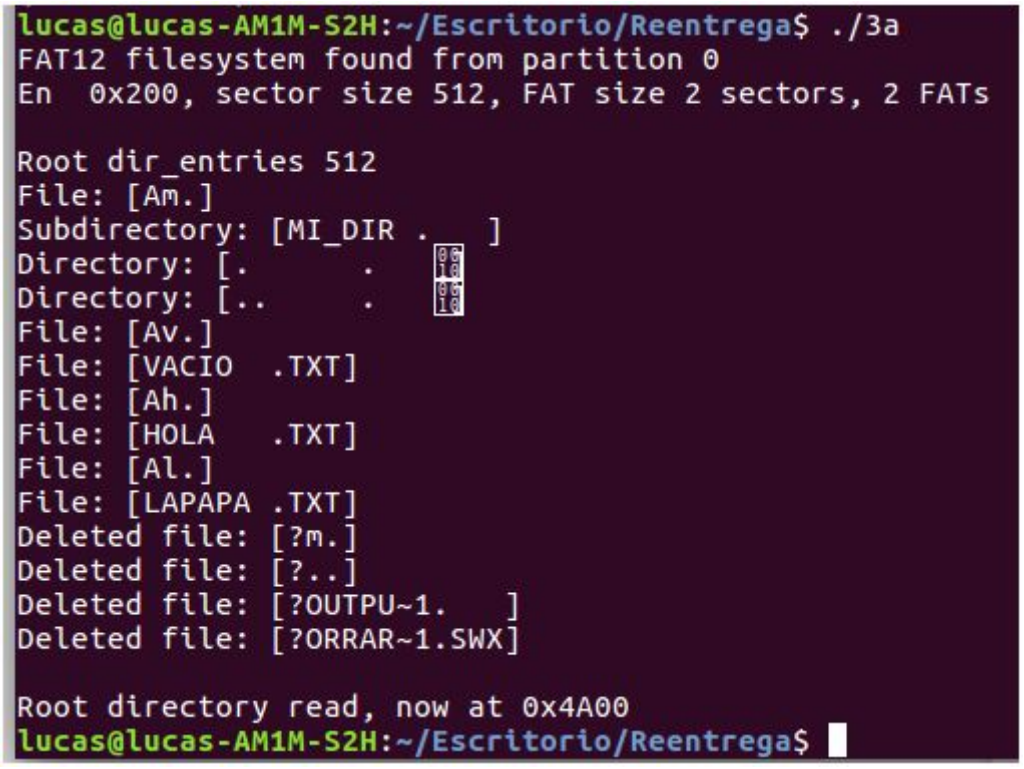

Fig. 15: Listado de archivos con el archivo "lapapa.txt"

Con estos datos y la información en la tabla 4 podemos ubicar el contenido del archivo con Bless, como se ve en la figura 16. Luego, podemos generar código para mostrarlo por pantalla, como se ve en la figura 17.

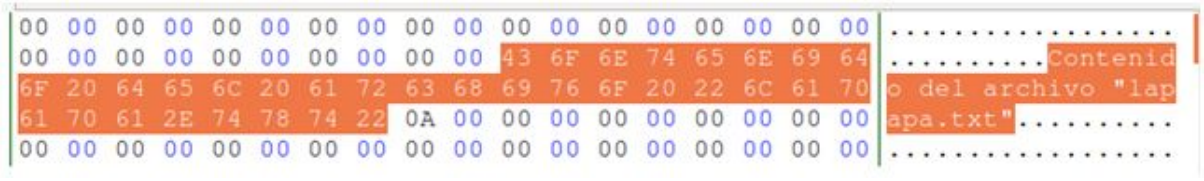

Fig. 16: Contenido del archivo "lapapa.txt", en hexadecimal y en ASCII.

Fig. 17: Contenido del archivo “lapapa.txt”, por código y en consola.