Administrador de File System FAT16 y FAT32

Julio Peralta, Lorena Ibaez, Mariano A. Carabajal,
Matias D. Rotela, Patricia Gómez
Dpto. de Ingeniería e Investigaciones Tecnológicas, Universidad Nacional de La Matanza,
Florencio Varela 1903 C.P. 1754 San Justo, Argentina
{julioperalta273, ib_lorenita, carabajalmariano, matyasdr,
patriciagomez86}@yahoo.com.ar,

Resumen. El presente trabajo expone una investigación y un análisis sobre los sistemas de archivos FAT16 y FAT32. Luego se describirá como fueron implementados los citados sistemas de archivos en el sistema operativo SODIUM, se explicarán las principales estructuras utilizadas, se detallará la función principal de cada una de las llamadas al sistema (syscalls) creadas para nuestro trabajo. Además, por último, se mencionará la forma de probar los distintos comandos implementados en el sistema operativo SODIUM.

Palabras Clave: sistema de archivos, FAT16, FAT32, SODIUM, syscalls, BPB.

1 Introducción

El propósito principal de este documento es mencionar brevemente las características de los sistemas de archivos FAT16 y FAT32, esta investigación fue necesaria para poder realizar la implementación de dichos sistemas de archivos en SODIUM. Además se describirá la implementación de FAT16 y FAT32 en el citado sistema operativo.

2 Investigación

Para llevar a cabo la investigación consultamos libros, sitios webs, código libre acerca de la implementación de los sistemas de archivos FAT16 y FAT32.
Un sistema de gestión de archivos es aquel sistema de software que proporciona a los usuarios unos servicios relativos al empleo de archivo, esto acaba con la necesidad para el usuario o programador de desarrollar software de propósito especifico para cada aplicación. [1]
En 1987 apareció lo que hoy se conoce como el formato FAT16. Se eliminó el contador de sectores de 16 bits. El tamaño de la partición ahora estaba limitado por la cuenta de sectores por clúster, que era de 8 bits. Esto obligaba a usar clusters de 32

Kbytes con los usuales 512 bytes por sector. Así que el límite definitivo de FAT16 se situó en los 2 gigabytes.
FAT32 Fue la respuesta para superar el límite de tamaño de FAT16 al mismo tiempo que se mantenía la compatibilidad con MS-DOS en modo real. Microsoft decidió implementar una nueva generación de FAT utilizando direcciones de cluster de 32 bits (aunque sólo 28 de esos bits se utilizaban realmente). [2]

2.1 Características de FAT16.

- MS-DOS, Windows 95, Windows 98, Windows NT, Windows 2000, y algunos sistemas operativos UNIX pueden usarla.

- Hay muchas herramientas disponibles para hacer frente a los problemas y recuperar los datos.

- Si existe un error de inicio, se puede iniciar el equipo con un disquete de arranque MS-DOS.

- Es eficiente, tanto en velocidad como en almacenamiento, en volúmenes menores de 256 MB.

- La carpeta raíz puede manejar un máximo de 512 entradas. El uso de nombres de archivo largos pueden reducir significativamente el número de entradas disponibles.

- FAT16 está limitado a 65.536 clusters, pero debido a ciertos clusters están reservados, tiene un límite práctico de 65.524. Si tanto el número máximo de clusters y su tamaño máximo (32 KB) se alcanzan, la unidad más grande se limita a 4 GB en Windows 2000. Para mantener la compatibilidad con MS-DOS, Windows 95 y Windows 98, un volumen FAT16 no debe ser mayor que 2 GB.

- El sector de arranque no está respaldado. [3]

2.2 Características de FAT32.

- FAT32 admite unidades de hasta 2 terabytes de tamaño.

- FAT32 aprovecha el espacio de forma más eficiente. FAT 32 utiliza clústeres menores (es decir, clústeres de 4 KB a 8 KB), lo que significa entre un 10 y un 15 por ciento de mejora en el uso del espacio con respecto a unidades grandes con sistemas de archivos FAT o FAT16.

- FAT32 es más robusto. FAT32 puede reubicar la carpeta raíz y utilizar la copia de seguridad de la tabla de asignación de archivos en lugar de la copia

predeterminada. Además, el registro de inicio de las unidades FAT32 se ha ampliado para incluir una copia de las estructuras de datos críticas. Por lo tanto, las unidades FAT32 son menos susceptibles a un único punto de error que las unidades FAT16 existentes.

- FAT32 es más flexible. La carpeta raíz de una unidad FAT32 es una cadena de clústeres ordinaria, de manera que puede ubicarse en cualquier unidad. Las limitaciones presentes en versiones anteriores con respecto al número de entradas de la carpeta raíz ya no existen. Además, se puede deshabilitar el duplicado de la tabla de asignación de archivos, con lo que se puede generar una tabla de asignación de archivos distinta de la primera que está activa. Estas características permiten el cambio de tamaño dinámico de las particiones FAT32. No obstante, hay que tener en cuenta que, aunque el diseño de FAT32 permite esta función, Microsoft no la implementará en la versión inicial. [4]

3 Implementación

Entre los objetivos fijados para la segunda parte del tp se encontraban la implementación de los sistemas de archivos FAT16 y FAT32. Para poder realizarlo hicimos uso de lo que ya habíamos implementado en la primera parte del tp, o sea, registramos los nuevos file system a nuestra capa de VFS. Para llevar a cabo esto es necesario seguir una serie de pasos.

3.1 Registrar un nuevo sistema de archivos en Sodium

Para registrar un nuevo sistema de archivos se debe agregar la información necesaria en el archivo vfs.c que se encuentra en la carpeta /kernel/fs. Ya que para que la capa de VFS pueda administrar la ejecución de los diferentes comandos es necesario que cuente con la información de los sistemas de archivos disponibles en el sistema operativo.

Con la implementación que proponemos para agregar un nuevo file system deberían seguirse los siguientes pasos:

1.- Crear implementación del manejo del file system que se desea agregar. Ejemplo fat16_fs.c, fat32_fs.c, etc.

2.- En el archivo vfs debe agregarse una función que permita cargar la información necesaria para el manejo de las operaciones del file system en el superbloque. Ejemplo

void vFnFat_CargarSBFAT16(stuSuperBloque *pstuSuperBloqueNuevo);

Esta función se utilizara para agregar en el superbloque la información del sistema de archivos real y poder cargar los punteros de las funciones a las operaciones correspondientes. Ejemplo

pstuSuperBloqueNuevo->s_op->eliminarDirectorio = iFnRmDir_FAT16;

3.- La dirección de la función que creamos en el punto anterior debe ser asociada al tipo de file system. Esto se logra añadiendo en la función vFnAgregarFileSystemType del archivo vfs.c las siguientes líneas

```
if(iFnCompararCadenas(stNombreTipoFS,<NOMBRE_NUEVO_FS>) == 1)
    {
        pstuFSTypeAux->cargar_SuperBloque                    =
<FUNCION_CARGA_INFO_BLOQUE>;
    }
```

4.- Agregar el tipo de file system a la lista de file systems registrados.

En la función vFnIniciarVFS que se encuentra en el archivo vfs.c agregar el nombre del file system que se quiere agregar.

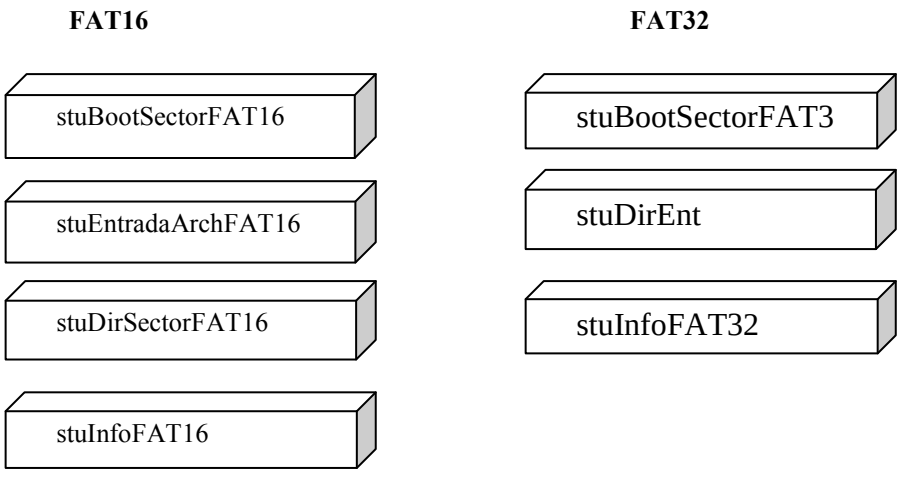
5.- En caso de necesitar nuevas operaciones para el manejo del file system, se deben agregar en la estructura stuSuperOperaciones.

Una vez registrado el file system se terminaron de definir y poner a prueba las estructuras de cada uno de los mismos

3.2 Análisis sobre la implementación

En base a la información que obtuvimos de nuestra investigación y al desarrollo anterior de FAT12 creamos las estructuras que consideramos necesarias para el manejo de FAT16 y FAT32.

Las estructuras implementadas son las siguientes:



FAT16

- **stuBootSector**: Contiene toda la información de BPB.

- **stuEntradaArchFAT16**: Estructura con la información del contenido de una entrada de directorio.
- **stuDirSectorFAT16**: Estructura para el manejo de directorios por sector.
- **stuInfoFAT16**: Es la estructura global de FAT16 y contiene la información del sector de arranque, las fats, el directorio raíz, y el directorio actual.

FAT32

- **stuBootSectorFAT32:** Contiene toda la información de BPB en FAT32.
- **stuDirEnt:** Estructura con la información del contenido de una entrada de directorio en FAT32.
- **stuInfoFAT32:** Es la estructura global de FAT32 y contiene la información del sector de arranque, las fats, el directorio raíz, y el directorio actual.

Para poder operar sobre los archivos se implemento en vfs la estructura stuFile, la misma cuenta con la información actual de cada archivo que se encuentra activo.

```
typedef struct stuFile
{
   int iFileDescriptor;
   char szPathCompleto[L_PATH];
   int iModoApertura;
   unsigned char *puszDirInicioArch;
   unsigned char *puszDirActualArch;
   int iTamanioArch;
   struct stuFile *pstuSiguiente;
} stuFile
```

Syscalls

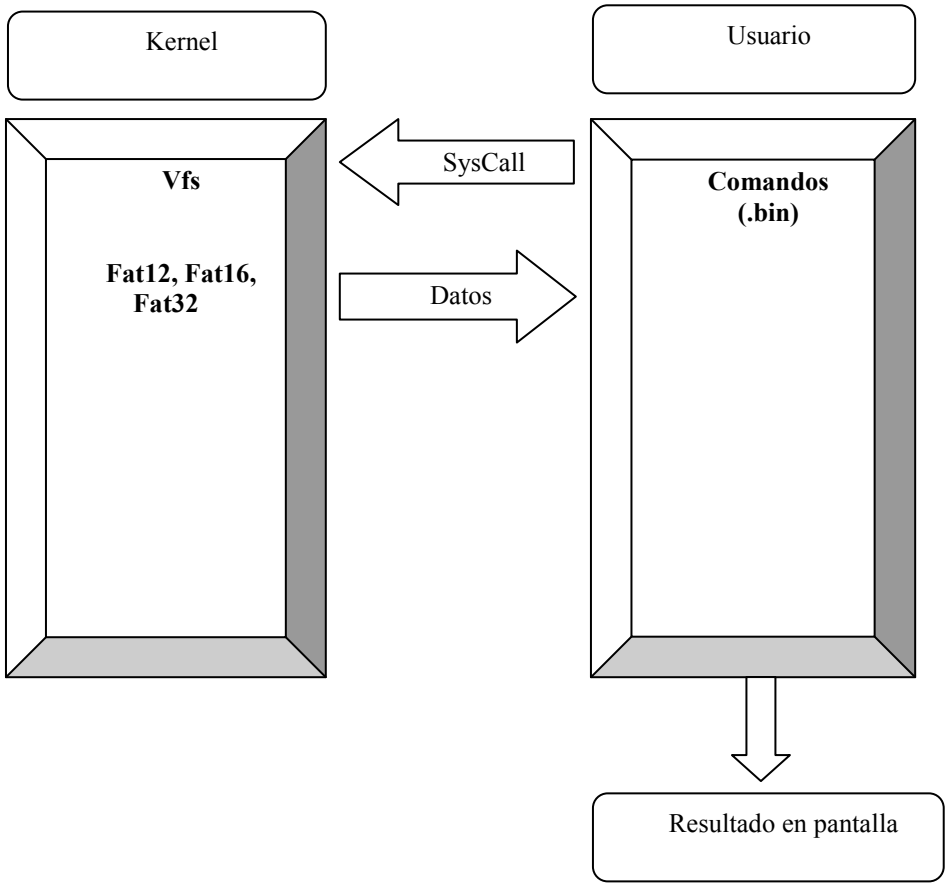
Int open_file (Nombre_archivo, Modo_ apertura)
Esta syscall me permite abrir un archivo, ya sea en fat12, fat16 o fat32.

Int read_file (Descriptor, Buffer, Bytes_a_ leer)
Esta syscall me permite leer los datos de un archivo, ya sea en fat12, fat16 o fat32.

Int write_file (Descriptor, Contenido_archivo, Tamaño_archivo)
Esta syscall me permite escribir los datos en un archivo, ya sea en fat12, fat16 o fat32.

Void close_file (Descriptor)
Esta syscall me permite cerrar un archivo, ya sea en fat12, fat16 o fat32.

A continuación se detalla una vista global de cómo quedaron implementados los sistemas de archivos FAT12, FAT16 y FAT32 en SODIUM:



3.3 Implementación del manejo de archivos con nombres largos (LFN) para FAT32

Al implementar fat32 se decidió que, además de contar con los comandos que se encontraban desarrollados previamente para fat12 y fat16, también soportara el uso de archivos con nombres largos. Para poder implementar esta funcionalidad fue necesario investigar como funcionaba la misma. Una de las fuentes de investigación fue el documento "Long Filename Specification" de Microsoft [5], el cual especifica el diseño para soportar archivos con nombres largos en el sistema de archivos FAT.

Unas de las limitaciones que tenían los sistemas de archivos FAT era no soportar nombres de archivos que superaran los 8 caracteres, para poder superar esta limitación se implementó el sistema VFAT. El cual no es un sistema de archivos en sí mismo, sino una especie de subsistema de archivos, que se puede colocar sobre un sistema de

archivos FAT12, FAT16 o FAT32. El sistema VFAT es una forma de ocultar los archivos de nombres largos en la estructura de directorios de los sistemas de archivos FAT. Por cada entrada de directorios que existe, se encuentra una o más entradas ocultas (entrada LFN) la cual permite almacenar los nombres largos.
Los nombres de archivo se almacenan utilizando caracteres Unicode que son de 16 bits de largo.

Dependiendo de la longitud del nombre de archivo largo, el sistema creará un número inválido de entradas en la tabla de directorios, estas son las entradas LFN. Estas entradas LFN se almacenan de manera que la primera que aparece es la que contiene la ultima parte del nombre y la primera se coloca sobre la entrada de directorio valida, la cual posee el nombre corto del archivo. En la siguiente imagen se muestra como se almacenan las entradas de directorios.

Last Long Name Directory Entry
. . .
2nd Long Name Directory Entry
1st Long Name Directory Entry
Existing 8.3 Directory Entry

Las entradas LFN contienen las siguientes características:
Tamaño: 32 bytes, con la siguiente estructura

Structure for LFN Entries

Byte Range	Description
0	Sequence Number and allocation Status
1 - 10	File Name Characters (Unicode)
11	File Attributes
12	Reserved
13	Checksum
14 - 25	File Name Characters (Unicode)
26 - 27	Reserved
28 - 31	File Name Characters (Unicode)

Atributos: siempre es 0x0F. Establece que los bits de volumen, sistema, solo lectura y oculto se encuentren seteados, por ende las entradas se pueden detectar y ocultar a la vista del usuario.

Número de Secuencia: Indica la secuencia de las entradas necesarias para cada archivo que contiene un nombre largo.

En sodium la estructura que maneja las entradas LFN es stuFatLfn y se encuentra definida en commonFat.

```
typedef struct stuFatLfn{
    char  cSeqno;
    char  cNombre_p1[10];
    char  cAttr;
    char  cReservado;
    char  cChksum;
    char  cNombre_p2[12];
    short sStart_cluster;
    char  cNombre_p3[4];
} NOALIGN stuFatLfn;
```

Las diferentes funciones que operan sobre estas estructuras también se encuentran definidas en commonFat, actualmente solo el sistemas de archivos FAT32 trabaja con nombres largos.

4 Pruebas

4.1 Pasos para poder realizar las pruebas de comandos en SODIUM

Al iniciarse SODIUM el sistema de archivos montado por defecto es FAT12, es decir que se pueden ejecutar los distintos comandos para FAT12, si se quiere montar otro sistema de archivos simplemente hay que ejecutar lo siguiente:

Montar [Sistema de Archivos]
Por ejemplo:
Montar FAT16

Luego hay que posicionarse en la unidad asignada, que la conocemos ejecutando el comando montar sin parámetros.

Por ejemplo:
Montar

A continuación realizamos cd [Unidad Asignada][Punto]
Por ejemplo:
Cd B.

De esta manera ya estamos posicionados en la unidad correspondiente donde se pueden ejecutar los distintos comandos.

Los comandos que disponemos en la actualidad son los siguientes: rm, rmdir, mkdir, touch, cd, ls, write, less, find, mv, cp, cat, dddisk, ls -r, find –r, cp –r.

Para conocer la forma de utilización de cada uno de los comandos solo basta con escribir en línea de comandos: nombre del comando -- ayuda.

4.2 Modificar FS desde Linux y luego levantarlo desde SODIUM

Pasos necesarios para crear la imagen en Linux:

1- Se crea la imagen
sudo dd if=/dev/zero of=floppy.img bs=1474560 count=1

2- Se le da el formato de FAT12
sudo /sbin/mkfs.vfat -v -F12 floppy.img

3- Se monta la imagen en el directorio /media/floppy2
mount -o loop -t vfat floppy.img /media/floppy2

4- Una vez montada la imagen creamos los directorios y archivos dentro de la imagen

5- Desmontamos la imagen de FAT12 con los directorios y archivos creados
umount /media/floppy2

6- Tomamos unicamente unos 60Kb de la imagen para luego levantarla en Sodium
sudo dd if=floppy.img of=imgFat12 bs=61440 count=1

7- Seteamos los permisos del archivo de imagen
chmod 777 imgFat12

8- Copiamos el archivo imgFat12 en el directorio TRUNK de Sodium y lo ejecutamos.

4.3 Grabar las modificaciones del FS en SODIUM y luego levantarlo en Linux

Al desmontar la imagen de FAT12 se sobrescribe el archivo sodium_fat12.img que se crea en la carpeta Build. Para ver las modificaciones desde linux se debe montar la imagen ejecutando:

En sodium:
Luego de haber realizado las diferentes operaciones sobre los archivos y directorios ejecutar los siguientes comandos para montar una nueva unidad y poder desmontar la unidad donde se hicieron los cambios.

>> montar FAT12B diskette

>> cd B.

>> desmontar A.

En Linux:

>>mount -o loop -t vfat sodium_fat12.img /media/floppy2

5 Conclusiones

Es interesante terminar nuestro trabajo mostrando las conclusiones o ideas más importantes.

Luego de haber investigado y trabajado en la implementación de los sistemas de archivos FAT16 y FAT32 en un sistema operativo real podemos decir que pudimos comprender las características más importantes de los citados sistemas de archivos, como ser tamaño de partición soportada, eficiencia en almacenamiento, de que manera funciona el manejo de nombres largos.

Además comprendimos la comunicación entre los distintos comandos que ejecutan en el shell de usuario y las funcionalidades de nivel kernel, que realizan efectivamente las operaciones, por medio de llamadas al sistema (syscalls).

Una de las cosas mas importantes a destacar fue la gran ayuda que nos represento haber implementado en una primera parte la capa de virtualización del file system, gracias a la misma nos fue posible implementar diferentes comandos a nivel usuario sin necesidad de modificar o agregar muchas líneas de código, para ser más especifico, en algunos casos con solo indicar la función del sistema de archivos que se encargaba de llevar a cabo el comando ya era suficiente, en otros ni era necesario modificar nada de la parte del kernel ya que el nuevo comando hacia uso de otros comandos ya definidos.

6 Referencias

[1]. William Stallings - Sistemas Operativos – Prentice Hall (1997) Edición N°2.
[2]. Wikipedia, http://es.wikipedia.org/wiki/Tabla_de_asignacion_de_archivos
[3]. Microsoft TechNet, http://technet.microsoft.com/en-us/library/cc940351.aspx
[4]. Microsoft Support, http://support.microsoft.com/kb/154997/es
[5]. Microsoft Corporation, Long Filename Specification