Resumen histórico de la Investigación

Antecedentes

En esta sección vamos a presentar dos de los documentos que armamos durante el transcurso de la investigación. Decidimos mostrarlos de esta manera, como dos documento, porque cada uno esta hecho en etapas distintas y de esta forma el lector puede ir viendo como nos fuimos adaptando a los cambios de la investigación y como estos impactaban en los resultados obtenidos.

En el primer documento llamado "Diferencias entre FAT16 y FAT32" presentamos una explicación detallada de las diferencias entre estos dos tipos de FAT's y además la estructura de FAT12, para luego hacer una comparación entre las tres. De esta forma, pudimos meternos más en tema y analizar bien como esa diferencia en las estructuras a nosotros nos iba a impactar en el momento de implementar FAT16 y FAT32. El lector podrá ver primero la estructura genérica de FAT, y luego el detalle de cómo esta formada cada parte. Todo este documento esta basado en la documentación original creada por Microsoft.

En el segundo documento llamado "Análisis.Funcionamiento.SODIUM" el lector podrá ver entre otras cosas:

1. la relación directa que existe entre el nombre de las variables que aparecen en el estándar de Microsoft y el nombre con el que aparecen en SODIUM
2. para que sirve cada una de esas variables
3. las estructuras que se usan en SODIUM para el manejo del sistema de archivos FAT
4. las funciones que utiliza
5. el panorama con el que nos encontramos al comienzo de la investigación.

También al final y previo a la conclusión que realizamos, se demuestra como se fue analizando lo que se iba encontrando y de la manera que creíamos que se podría solucionar.
A lo largo de los documentos van a ver como se iban cambiando los objetivos y los distintos enfoques que íbamos adoptando. Antes de realizar un cambio profundo, era necesario crear una buena base teórica y un punto de partida firme que pueda soportar dichos cambios.

Diferencias entre Fat16 y Fat32

Aldo Flores, Pablo Sassón, Pablo Groba, Claudio D'Amico
Universidad Nacional de La Matanza

Abstract
El propósito principal de este documento es aclarar las diferencias (sustanciales o no) entre los distintos tipos de FAT's, con el objetivo de apoyar al desarrollo e implementación del driver/modulo de lectura/escritura de este sistema de archivos para el Sistema Operativo SODIUM, desarrollado activamente en la institución. Actualmente hoy el SODIUM solo puede leer y escribir sobre FAT12, en caso de ser sobre diskette, o sobre FAT16, en caso de ser sobre disco rígido, así que es deseable tener un documento que arroje luz sobre la estructura de sistemas de archivos FAT32 para su futura implementación y cuya principal diferencia entre todos ellos se va a encontrar en el tamaño en bit's de cada entrada en la Tabla de Alocación de Archivos (FAT).

Palabras Clave
FAT, Cluster, Sector, SODIUM, FAT12, FAT16, FAT32, Boot Sector, etc.

Introducción

El propósito principal de este documento es aclarar las diferencias (sustanciales o no) entre los distintos tipos de FAT's sin ahondar en datos históricos. Se presupone que el lector ya cuenta con la noción de lo que significa un sector y/o cluster y además esta familiarizado con FAT12 y FAT16. Por esto solo profundizaremos en FAT32 y sus diferencias con sus dos FileSystems predecesores.

Breve Reseña de los diferentes tipos de FAT

FAT12

La versión inicial de FAT se conoce ahora como FAT12. Es un sistema de archivos para disquete, por lo que tiene varias limitaciones:

No soporta anidación de carpetas.
Las direcciones de bloque solamente contienen 12 bits. Esto complica la implementación.
El tamaño del disco se almacena como una cuenta de 16 bits expresada en sectores, lo que limita el espacio manejable a 32 megabytes.

FAT16

En el sistema de archivos FAT16 las direcciones de clúster no pueden ser mayores a 16 bits. El número máximo de clusters al que se puede hacer referencia con el sistema FAT es, por consiguiente, 2^16 (65536) clusters. Ahora bien, ya que un clúster se compone de un número fijo (4, 8, 16, 32,...) de sectores de 512 bytes contiguos, el tamaño máximo de la partición FAT se puede determinar multiplicando el número de clusters por el tamaño de un clúster.

Limitaciones

Es más probable encontrarse con problemas de creación de archivos o carpetas en el directorio raíz, ya que FAT16 sólo asigna espacio para 512 entradas de directorio raíz. Debido a que si usamos nombres de archivo largos, podemos ocupar más de una entrada de directorio, se puede tener menos de 512 archivos o carpetas en el directorio raíz. De hecho hay espacio sólo para 25 nombres de archivo largos de longitud máxima (512/20).

FAT32

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).**
En teoría, esto debería permitir aproximadamente 268.435.538 clusters, arrojando tamaños de almacenamiento cercanos a los ocho terabytes. Sin embargo, debido a limitaciones en la utilidad ScanDisk de Microsoft, no se permite que FAT32 crezca más allá de 4.177.920 clusters por partición (es decir, unos 124 gigabytes). Posteriormente, Windows 2000 y XP situaron el límite de FAT32 en los 32 gigabytes. Microsoft afirma que es una decisión de diseño, sin embargo, es capaz de leer particiones mayores creadas por otros medios.
El tamaño máximo de un archivo en FAT32 es 4 gigabytes (2^32−1 bytes), lo que resulta engorroso para aplicaciones de captura y edición de video, ya que los archivos generados por éstas superan fácilmente ese límite.
Para solucionar este problema, FAT32 utiliza un direccionamiento de cluster de 32bits, lo que en teoría podría permitir manejar particiones cercanas a los 2 Tib (Terabytes), pero en la práctica Microsoft limitó estas en un primer momento a unos 124Gb como se explicó anteriormente, **fijando posteriormente el tamaño máximo de una partición en FAT32 en 32Gb**. Esto se debe más que nada a una serie de limitaciones del Scandisk de Microsoft, ya que FAT32 puede manejar particiones mayores creadas con programas de otros fabricantes. Un claro ejemplo de esto lo tenemos en los discos externos multimedia, que están formateados en FAT32 a pesar de

ser particiones de bastante tamaño (en muchos casos más de 300Gb).

El tamaño del cluster utilizado sigue siendo de 32Kb, lo que sigue significando un importante desperdicio de disco, ya que un archivo de 1Kb (que los hay, y muchos además) está ocupando en realidad 32Kb de disco.

-Para calcular el tamaño del cluster, podremos hacer:

Tamaño del Cluster = Capacidad del Disco / Número posible de clusters.

Y ya que el tamaño del cluster es directamente proporcional al espacio desperdiciado (en otras palabras, cuando crece el tamaño del cluster, el desperdicio aumenta), podemos notar que queremos un sistema de archivos que pueda manejar una gran cantidad de clusters. Y es aquí en donde se diferencian el FAT32 y el FAT16.

Diferencias Básicas

El FAT16 usa 16 bits para contar los clusters. Es decir lo máximo que puede contar es hasta 2^16 - 1, es decir, hasta 65535. Es decir, lo máximo que puede caber son 65535 clusters. Por eso, a medida que aumenta el tamaño de tu disco duro, aumentará también el tamaño de los clusters, ya que el número máximo es el citado.

También cada sector dentro de un grupo (cluster) debe ser numerado. Cada sector tiene un número de índice que está en un byte (es decir, 8 bits). Pero se utilizan solamente siete de estos bits, por lo tanto, el número máximo de sectores en cada cluster es de 128.

Calculando, tendremos:

Un máximo de 65535 clusters
Un máximo de 128 sectores por cluster
512 bytes por sector.

Entonces, el tamaño máximo de FAT16 es = 65535 * 128 * 512 = 4 GB

Pero si consideramos que 128 * 512 bytes es 64K, para lo cual necesitamos más que 16 bits, entonces tendremos que utilizar este tope, es decir, sacar un BIT a esta suma, resultando por tanto en 32KB como máximo para cada cluster.

Así, recalculando, tenemos:

Tamaño máximo en FAT16 = 65535 * 32KB = 2 GB.

El FAT 32 resuelve este problema, ya que aumenta el límite máximo de clusters que puede manejar, usando 32 bits.

Otras diferencias entre ellos son:

En la FAT32, el directorio raíz tiene tamaño ilimitado. Esto significa que puede haber cualquier cantidad de archivos en el directorio raíz. En la FAT16, el máximo era de 255 archivos en la raíz. La FAT32 tiene un sistema de redundancia mejor. Ambos sistemas guardan dos copias de la FAT en el disco. Pero en la FAT32 el sistema puede elegir leer de cualquiera de ellas, lo que da mayor tolerancia a fallas que podrían ocurrir por tablas corrompidas.

	FAT12	FAT16	FAT32
Desarrollador	Microsoft		
Nombre completo	Tabla de Asignación de Archivos		
Identificador de partición	0x01 (MBR)	0x04, 0x06, 0x0E (MBR)	0x0B, 0x0C (MBR)
Contenido de carpeta	Tabla		
Ubicación de archivo	Lista enlazada		
Bloques defectuosos	Lista enlazada		
Tamaño máximo de archivo	32MB	2GB	4GB
Número máximo de archivos	4077	65517	268.435.437
Longitud máxima del nombre de archivo	8.3 (11) o 255 caracteres cuando se usan LFNs (*Long File Names*)		
Tamaño máximo del volumen	32MB	2GB	2TB
Fechas almacenadas	Creación, modificación, acceso		
Rango de fechas soportado	1 de enero de 1980 – 31 de diciembre de 2107		

Tabla 1 - Diferencias entre

Ventajas y Desventajas

	Ventajas	Desventajas
Fat16	•MS-DOS, Windows 95, Windows 98, Windows NT, Windows 2000, y algunos sistemas operativos UNIX pueden usarlo. • Hay muchas herramientas disponibles para resolver los problemas y recuperar datos. • Si tiene un fallo de inicio, puede iniciar el equipo con un disquete de disco de arranque MS-DOS. • Es eficiente, tanto en velocidad y almacenamiento, sobre los volúmenes más pequeños de 256 MB.	• La carpeta raíz puede manejar un máximo de 512 entradas. El uso de nombres de archivo largos pueden reducir el número de entradas disponibles. • FAT16 está limitado a 65.536 clústeres, pero debido a ciertos grupos son reservados, tiene un límite práctico de 65.524. Si se alcanzan tanto el número máximo de grupos y su tamaño máximo (32 KB), la unidad más grande está limitada a 4 GB en Win2k. • El sector de arranque no tiene una copia de seguridad. • No hay un plus de seguridad o sistema de archivos de compresión de archivos con FAT16. • FAT16 puede desperdiciar el espacio de almacenamiento de archivos en unidades más grandes tanto como el tamaño del clúster se agrande. El espacio asignado para almacenar un archivo se basa en el tamaño de la granularidad de asignación de clúster, no del tamaño del archivo. Un 10-KB archivo almacenado en un clúster de 32 KB genera 22 KB de espacio desperdiciado en disco.
Fat32	FAT32 asigna espacio en disco mucho más eficiente que las versiones anteriores de FAT. Dependiendo del tamaño de los archivos, hay un potencial de decenas e incluso cientos de megabytes de espacio en disco libre en las grandes unidades de disco duro. Además, FAT32 presenta las siguientes mejoras: • La carpeta raíz en una unidad FAT32 ahora es una cadena de clústeres común, por lo que se puede localizar en cualquier instancia de parte del volumen. Por esta razón, FAT32 no limita el número de entradas en la carpeta raíz. • Utiliza el espacio más eficientemente que FAT16. FAT32 utiliza clusters más pequeños (4 KB para unidades de hasta 8 GB), lo que resulta en 10 a 15 por ciento el uso más eficiente del espacio en relación a las grandes unidades FAT16. FAT32 también reduce los recursos necesarios para que la PC opere. • FAT32 es más robusto que FAT16. FAT32 tiene la capacidad de reubicar el directorio raíz y utilizar la copia de seguridad de la FAT en lugar de la copia predeterminada. Además, el registro de arranque de las unidades FAT32 se ha ampliado para incluir una copia de seguridad de las estructuras de datos críticos. Esto significa que los volúmenes FAT32 son menos susceptibles a un único punto de fallo de volúmenes FAT16.	• El mayor volumen al cual Windows 2000 puede dar formato en FAT 32 tiene un tamaño limitado a 32 GB. •Los volúmenes FAT32 no son accesibles desde cualquier otro sistema operativo que no sea Windows 95 OSR2 y Windows 98. •El sector de arranque no tiene una copia de seguridad. •No tiene incorporada seguridad o sistema de compresión de archivo con FAT32.

Tabla 2 – Cuadro Sinóptico de Ventajas Y Desventajas

Diseño de un FileSystem FAT32:

En un sistema FAT32 el espacio físico de almacenamiento se organiza en 3 grandes partes:

- Región reservada:
- BPB Bios Parameter Block
- Estructura File System Info
- Copia de la BPB Bios Parameter Block (opcional aunque casi siempre presente)

- Región de las FATs:
- FAT
- Copia de la FAT (opcional aunque casi siempre presente)

- Región de datos de directorios y ficheros

Región reservada:

Nosotros para enfocarnos a nivel registro, vamos a hacer un análisis de la primera sección, la Región reservada, mas precisamente BPB Bios Parameter Block, o lo que conocemos como Boot Sector. La región reservada comienza en el sector 0 de la partición, que no en el sector 0 del soporte, ya que algunos sistemas cuentan con un MBR que esta ubicado en dicha posición. El BPB Bios Parameter Block o Boot Sector es el primer sector del soporte y contiene información elemental del dispositivo. El mismo se divide en dos partes. La primer parte es similar para FAT32 como también para FAT12 y FAT16, diferenciándose solo en la segunda parte

Nombre Microsoft	Offset	Tamaño	Descripción	Nombre Sodium	Valor Sodium
BS_jmpBoot	00h (00)	3 Bytes	Instrucción de salto al código de arranque. Es la forma de una instrucción de salto de tres bytes para Intel x86 que salta al principio del código de arranque del sistema operativo.		
BS_OEMName	03h	8 Bytes	OEM Name. Cadena que identifica el fabricante del disco.	strOemId	″Sodium″
BPB_BytsPerSec	0Bh	1 Word	Bytes por Sector.	dwBytesPorSector	0x0200
BPB_SecPerClus	0Dh	1 Byte	Sectors Por Cluster.	dbSectoresPorCluster	0x01
BPB_RsvdSecCnt	0Eh	1 Word	Sectores Reservados.	dwSectoresReservados	0x0001
BPB_NumFATs	10h	1 Byte	Numeros de copias de FAT.	dbTotalFATs	0x02
BPB_RootEntCnt	11h	1 Word	Maxima cantidad de entradas en el directorio raíz.	dwEntradasRaiz	0x00E0
BPB_TotSec16	13h	1 Word	Numero de Sectores en una partición menor a 32MB.	dwTotalSectores	0x0B40
BPB_Media	15h	1 Byte	Media Descriptor (F8h for Hard Disks).	dbDescriptor	0xF0
BPB_FatSz16	16h	1 Word	Sectores Por FAT.	dwSectoresPorFat	0x09
BPB_SecPerTrk	18h	1 Word	Sectores Por Pista.	dwSectoresPorPista	0x12
BPB_NumHeads	1Ah	1 Word	Numero de cabezas.	dwNroCabezas	0x02
BPB_HiddSec	1Ch	1 Double Word	Numeros de sectores ocultos en la partición.	dbSectoresOcultos	0x00000000
BPB_TotSec32	20h	1 Double Word	Numero de sectores en la partición. Se utiliza en FAT32. También en FAT12/FAT16 se utilizan almacenar el número de sectores cuando este es mayor a 0x10000.	ddNroSectores	0x00000000

Tabla 3 – Primera Parte de la Región Reservada

Vamos a analizar esta segunda parte, que comienza en el byte 36 del sector 0 para FAT16:

Nombre Microsoft	Offset (byte)	Tamaño (bytes)	Descripción	Nombre Sodium	Valor Sodium
BS_DrvNum	64	1	Número de unidad para la Int 0x13 (por ejemplo, 0x80). Este campo admite arranque de MS-DOS y se establece en INT 0x13 el número de unidad de los medios de comunicación (0x00 para disquetes, 0x80 para los discos duros). NOTA: Este campo es en realidad específico para el sistema operativo.	dbNroDispositivo	0x00
BS_Reserved1	65	1	Reservado (Usado por Windows NT). El código que formatea los volúmenes de FAT32 siempre debe setear los bytes de este campo en 0.	dbFlags	0x00
BS_BootSig	66	1	Firma extendida de arranque (0x29). Esta firma es un byte que indica que los siguientes tres campos en el sector de arranque están presentes.	dbSignature	0x29
BS_VolID	67	4	Numero serial del volumen. Este campo, junto con BS_VolLab, soporta el seguimiento de volumen en un medio extraíble. Estos valores permiten a los controladores de sistema de archivos FAT poder detectar que un disco equivocado se inserta en una unidad extraíble. Este identificador se suele conseguir simplemente combinando la fecha y hora actuales en un valor de 32-bit.	ddIdVolumen	0x0FFFFFFF
BS_VolLab	71	11	Etiqueta del volumen. Este campo coincide con la etiqueta de volumen 11-byte registrados en el directorio raíz.	strNombreVolumen	"SODIUM BOOT"
BS_FilSysType	82	8	Siempre seteado con el string " FAT32 ". NOTA: Mucha gente piensa que el string de este campo tiene algo que ver con la determinación de qué tipo de FAT se está utilizando, si FAT 12-16-32. Esto no es verdadero. Este campo es usado solo a nivel informativo, no es usado para determinar el tipo de FAT porque frecuentemente no es correctamente seteado o no está presente.	strTipoFS	"FAT12 "

Vamos a analizar esta segunda parte, que comienza en el byte 36 del sector 0 para FAT32:

Nombre	Offset (byte)	Tamaño (bytes)	Descripción
BPB_FATSz32	36	4	Este campo solo está definido para FAT32 y no existe para FAT16 ni para FAT 12. Este campo es el contador de los sectores ocupados por una FAT. BPB_FATSz16 debe ser 0.
BPB_ExtFlags	40	2	Este campo solo está definido para FAT32 y no existe para FAT16 ni para FAT 12. Bits 0-3: Número de FAT activa en base cero. Sólo es válido si el mirroring esta deshabilitado. Bits 4-6: Reservado. Bit 7: 0 significa que la FAT se refleja en tiempo de ejecución en todas las FATs. 1 significa que solo una FAT esta activa; es la que se hace referencia en bits 0-3. Bits 8-15: Reservado.
BPB_FSVer	42	2	Este campo solo está definido para FAT32 y no existe para FAT16 ni para FAT 12. El byte alto es el número de revisión mayor. El byte bajo es el número de revisión menor. Este es el número de versión del volumen FAT. Esto soporta la posibilidad de extender el tipo de soporte FAT32 en el futuro sin preocuparse por los viejos drivers de FAT32 a la hora de montar el volumen. Si este campo no es cero, el nuevo nível de versiones de Windows no montará el volumen.
BPB_RootClus	44	4	Este campo solo está definido para FAT32 y no existe para FAT16 ni para FAT 12. Este es seteado con el primer cluster del directorio root, usualmente con 2, pero no necesariamente debe ser 2.
BPB_FSInfo	48	2	Este campo solo esta definido para FAT32 y no existe para FAT16 ni para FAT 12. El número de sector de la estructura FSINFO es un área reservada del volumen FAT32. Usualmente es 1.
BPB_BkBootSec	50	2	Este campo solo está definido para FAT32 y no existe para FAT16 ni para FAT 12. Si no es cero, indica el número del sector en el área reservada del volumen, donde se ubica una copia del boot record. Usualmente 6. No se recomienda otro valor que no sea 6.
BPB_Reserved	52	12	Este campo solo está definido para FAT32 y no existe para FAT16 ni para FAT 12. Es una reserva para una futura expansión. El código que formatea los volúmenes de FAT32 siempre debe setear los bytes de este campo en 0.
BS_DrvNum	64	1	Este campo tiene la misma definición tanto en FAT32 como en FAT 16 y FAT12. La única diferencia es que en FAT32 el campo está en un Offset diferente en el boot sector. Número de unidad para la Int 0x13 (por ejemplo, 0x80). Este campo admite arranque de MS-DOS y se establece en INT 0x13 el número de unidad de los medios de comunicación (0x00 para disquetes, 0x80 para los discos duros). NOTA: Este campo es en realidad específico para el sistema operativo.
BS_Reserved1	65	1	Este campo tiene la misma definición tanto en FAT32 como en FAT 16 y FAT12. La única diferencia es que en FAT32 el campo está en un Offset diferente en el boot sector. Reservado (Usado por Windows NT). El código que formatea los volúmenes de FAT32 siempre debe setear los bytes de este campo en 0.

BS_BootSig	66	1	Este campo tiene la misma definición tanto en FAT32 como en FAT 16 y FAT12. La única diferencia es que en FAT32 el campo está en un Offset diferente en el boot sector. Firma extendida de arranque (0x29). Esta firma es un byte que indica que los siguientes tres campos en el sector de arranque están presentes.
BS_VolID	67	4	Este campo tiene la misma definición tanto en FAT32 como en FAT 16 y FAT12. La única diferencia es que en FAT32 el campo está en un Offset diferente en el boot sector. Numero serial del volumen. Este campo, junto con BS_VolLab, soporta el seguimiento de volumen en un medio extraíble. Estos valores permiten a los controladores de sistema de archivos FAT poder detectar que un disco equivocado se inserta en una unidad extraíble. Este identificador se suele conseguir simplemente combinando la fecha y hora actuales en un valor de 32-bit.
BS_VolLab	71	11	Este campo tiene la misma definición tanto en FAT32 como en FAT 16 y FAT12. La única diferencia es que en FAT32 el campo está en un Offset diferente en el boot sector. Etiqueta del volumen. Este campo coincide con la etiqueta de volumen 11-byte registrados en el directorio raíz.
BS_FilSysType	82	8	Siempre seteado con el string " FAT32 ". NOTA: Mucha gente piensa que el string de este campo tiene algo que ver con la determinación de qué tipo de FAT se está utilizando, si FAT 12-16-32. Esto no es verdadero. Este campo es usado solo a nivel informativo, no es usado para determinar el tipo de FAT porque frecuentemente no es correctamente seteado o no está presente.

Tabla 4 – Segunda Parte de la Region Reservada

FS Information Sector (Sector con información del sistema de archivos)

Este sector fue introducido en FAT32 para aumentar la velocidad de acceso de algunas operaciones (en particular, obtener la cantidad de espacio libre). Esta estructura esta ubicada en el sector indicado en el boot record en la posición 0x30 (usualmente el sector 1, inmediatamente después del boot record).

Byte Offset	Tamaño (bytes)	Description
0x00	4	Valor que indica que este es el sector FSInfo (0x52 0x52 0x61 0x41 / "RRaA")
0x04	480	Reservado (0x00)
0x1e4	4	Otro valor que indica que este es el sector FSInfo (0x72 0x72 0x41 0x61 / "rrAa")
0x1e8	4	Números de cluster libres en el volumen, o -1 si es desconocido
0x1ec	4	Numero del cluster mas recientemente usado
0x1f0	14	Reservado (0x00)
0x1fe	2	Valor que indica que este es el final del sector (0x55 0xAA)

Tabla 5 – Sector con información del sistema de archivos

File Allocation Table (FAT - Tabla de asignación de archivos)

La tabla de asignación de archivos (FAT) es una lista de entradas que se asignan a cada cluster en la partición. Cada entrada registra una de las cinco cosas:

- el número de clúster del clúster siguiente en una cadena
- un carácter especial (EOC) que indica el final de una cadena
- una entrada especial para indicar un clúster no válido
- una entrada especial para indicar un clúster reservados
- un cero para indicar que el grupo no se utiliza

Cada versión del sistema de archivos FAT utiliza un tamaño diferente para las entradas FAT. En una tabla FAT más pequeña, el espacio desperdiciado en particiones grandes aumenta. El sistema de archivos FAT12 utiliza 12 bits por entrada FAT, por tanto, dos entradas ocupan 3 bytes. Utiliza el sistema 'little-endian': si se consideran

los 3 bytes como un número de 24 bits 'little-endian', los 12 bits menos significativos son la primera entrada y los 12 bits más significativos es la segunda.

FAT12	FAT16	FAT32	Description
0x000	0x0000	0x00000000	Cluster Libre
0x001	0x0001	0x00000001	Valor reservado; no se usa
0x002-0xFEF	0x0002-0xFFEF	0x00000002-0x0FFFFFEF	Cluster usado; el valor apunta al siguiente cluster
0xFF0-0xFF6	0xFFF0-0xFFF6	0x0FFFFFF0-0x0FFFFFF6	Valores reservados; no se usan
0xFF7	0xFFF7	0x0FFFFFF7	Sector defectuoso o cluster reservado
0xFF8-0xFFF	0xFFF8-0xFFFF	0x0FFFFFF8-0x0FFFFFFF	Ultimo cluster del archivo

Tabla 6 – FAT valores de entrada

Tenga en cuenta que FAT32 sólo utiliza 28 bits de los 32 bits posibles. Los 4 bits superiores son por lo general cero (como se indica en la tabla de arriba), están reservados y deben ser dejados intactos.
El primer cluster de la región de datos es el número 2. Eso deja a las dos primeras entradas de la FAT no utilizadas. En el primer byte de la primera entrada se almacena una copia del descriptor de medios de comunicación. Los restantes 8 bits (si es FAT16), o 20 bits (si es FAT32) de esta entrada son 1. En la segunda entrada el marcador se almacena el final de clúster (EOC). Los dos bits de máximo orden de la segunda entrada, en el caso de FAT16 y FAT32, a veces es utilizado para la gestión de errores volúmenes: un 1 en el bit de orden superior: último apagado del sistema fue limpio (sin errores); el siguiente bit de mayor orden en 1: durante el montaje anterior, fue detectado un error de ausencia de disco I/O.
En resumen, la Tabla de Asignación de Archivos es una lista de valores digitales que describe la asignación de los clusters de una partición o, dicho de otra forma, el estado de cada clúster de la partición en la que se encuentra. De hecho, cada célula de la tabla de asignación corresponde a un clúster. Cada célula contiene un número que indica si un archivo está utilizando el clúster. De ser así, indica la ubicación del siguiente clúster en el archivo. De esta forma, se obtiene una cadena FAT, la cual es una lista vinculada de referencias que apunta a los clusters sucesivos hasta el final del archivo. Cada entrada FAT tiene una extensión de 16 ó 32 bits (todo depende de si es una entrada FAT16 o FAT32). Las primeras dos entradas almacenan información acerca de la tabla misma, mientras que las entradas siguientes hacen referencia a los clusters. En realidad, cada partición contiene dos copias de la tabla almacenada de manera contigua en el disco, para que pueda recuperarse si la primera copia se corrompe.

El directorio raíz

Este índice es un tipo especial de archivo que almacena las sub-carpetas y archivos que componen cada carpeta. Cada entrada del directorio contiene el nombre del archivo o carpeta (máximo 8 caracteres), su extensión (máximo 3 caracteres), sus atributos (archivo, carpeta, oculto, del sistema, o volumen), la fecha y hora de creación, la dirección del primer cluster donde están los datos, y por último, el tamaño que ocupa.
El directorio raíz ocupa una posición concreta en el sistema de archivos, pero los índices de otras carpetas ocupan la zona de datos como cualquier otro archivo.
Los nombres largos se almacenan ocupando varias entradas en el índice para el mismo archivo o carpeta.

Conclusión

La estructura del sistema de archivos FAT32 es similar a FAT12 y FAT!6, ya implementadas en nuestro sistema operativo SODIUM. La principal diferencia es la cantidad de espacio que cada uno puede almacenar y administrar eficientemente. Al permitir mayor cantidad de clúster a mismo tamaño de volumen, FAT32 disminuye el espacio desperdiciado producto de la fragmentación interna porque el tamaño del cluster disminuye. FAT32 además presenta información sobre el estado actual del sistema de archivos, por ejemplo número de cluster libres y cual es el último cluster utilizado. Otra diferencia sustancial es que el directorio raíz no tiene entradas limitadas en FAT32 como si las tiene en FAT16.

Referencias

[1] http://es.wikipedia.org/wiki/Tabla_de_Asignación_de_Archivos
[2] http://www.microsoft.com/hwdev/download/hardware/fatgen103.doc
[3] http://www.configurarequipos.com/doc563.html
[4] http://es.kioskea.net/contents/repar/fat32.php3
[5] http://technet.microsoft.com/en-us/library/cc940351.aspx
[6] http://en.wikipedia.org/wiki/File_Allocation_Table

Análisis del funcionamiento SODIUM respecto a Fat16

Aldo Flores, Pablo Sassón, Pablo Groba, Claudio D'Amico
Universidad Nacional de La Matanza

Abstract
El propósito principal de este documento es comenzar a documentar el analisis realizado al sistema de archivos de SODIUM. Dicho sistema de archivo es FAT 16. Vamos a especificar tanto el nombre de las diferentes estructuras y valores de campos que se utilizan para desarrollar dicho sistema de archivo, como así también, en que archivo se encuentran dentro de la estructura del código del SODIUM. En la segunda parte del documento, se presenta el estado de nuestra investigación y los pasos a seguir en el futuro.

PARTE 1: Nombres de estructuras y valores en SODIUM

Nombres de campo y valores de la región reservada en el SODIUM:

La región reservada se divide en dos partes: la primera desde el byte 0 hasta el 36 del sector 0 y la segunda a partir del byte 36 de dicho sector.
La primer parte es similar para FAT32 como también para FAT12 y FAT16, diferenciándose solo en la segunda parte. Por lo tanto esta parte la podemos seguir utilizando cuando implementemos el sistema de archivo FAT32.

Nombre Microsoft	Nombre Sodium	Valor Sodium
BS_jmpBoot		
BS_OEMName	strOemId	"Sodium "
BPB_BytsPerSec	dwBytesPorSector	0x0200
BPB_SecPerClus	dbSectoresPorCluster	0x01
BPB_RsvdSecCnt	dwSectoresReservados	0x0001
BPB_NumFATs	dbTotalFATs	0x02
BPB_RootEntCnt	dwEntradasRaiz	0x00E0
BPB_TotSec16	dwTotalSectores	0x0B40
BPB_Media	dbDescriptor	0xF0
BPB_FatSz16	dwSectoresPorFat	0x09
BPB_SecPerTrk	dwSectoresPorPista	0x12
BPB_NumHeads	dwNroCabezas	0x02
BPB_HiddSec	dbSectoresOcultos	0x00000000
BPB_TotSec32	ddNroSectores	0x00000000

Tabla 7 – Primera Parte de la Región Reservada

Vamos a analizar esta segunda parte, que comienza en el byte 36 del sector 0 para FAT16. Si bien esta parte de la región reservada no estará presente cuando implementemos FAT32, al menos no de ésta forma, analizarla nos sirve para entender e interpretar las funciones y estructuras utilizadas en el sistema de archivos presente actualmente en el SODIUM:

Nombre Microsoft	Nombre Sodium	Valor Sodium
BS_DrvNum	dbNroDispositivo	0x00
BS_Reserved1	dbFlags	0x00
BS_BootSig	dbSignature	0x29
BS_VolID	ddIdVolumen	0x0FFFFFFF
BS_VolLab	strNombreVolumen	"SODIUM BOOT"
BS_FilSysType	strTipoFS	"FAT12 "

A continuación presentamos la estructura que tendrá la segunda parte de la región reservada, que comienza en el byte 36 del sector 0 para FAT32. Cuando empecemos con la implementación, se elegirán los nombres para estos campos y sus valores correspondientes:

Nombre Microsoft	Nombre Sodium	Valor Sodium
BPB_FATSz32		
BPB_ExtFlags		
BPB_FSVer		
BPB_RootClus		
BPB_FSInfo		
BPB_BkBootSec		
BPB_Reserved		
BS_DrvNum		
BS_Reserved1		
BS_BootSig		
BS_VolID		
BS_VolLab		
BS_FilSysType		

Tabla 8 – Segunda Parte de la Region Reservada para FAT32

Para poder manejar la información de la región reservada, en el SODIUM se utiliza una estructura denominada 'stuFsEncabezadoFAT'. Esta estructura es cargada en el SODIUM con datos del primer sector del disco, describe un file system FAT (cualquier versión):
Ubicación: **\sodium\include\fs\fat.h**

Codificación:

```
typedef struct stuFsEncabezadoFAT
{
	uint32_t ignored:24;			//3 bytes
	unsigned char oem_id[ 8 ];		//8 bytes
	uint16_t bytes_per_sector;		//2 bytes - 512
	uint8_t sectors_per_cluster;		//1 bytes - 1
	uint16_t reserved_sectors;		//2 bytes - 1
	uint8_t fats;				//1 bytes //cantidad de fats generalmente 2).
	uint16_t root_entries;			//2 bytes - 244 //Sector de comienzo del Directorio raiz
	uint16_t small_sectors;			//2 bytes - 2880 (Sectores totales)
	uint8_t media_descriptor;		//1 bytes - 240
	uint16_t sectors_per_fat;		//2 bytes - 9
	uint16_t sectors_per_track;		//8 bytes - 12 (sectores por pista)
	uint16_t heads;				//2 bytes - 2 (numero de caras)
	uint32_t hidden_sectors;		//4 bytes - 0
	uint32_t large_sectors;			//4 bytes 0
	uint8_t physical_drive_number;		//1 bytes -0
	uint8_t current_head;
	uint8_t signature;			//0x28 o 0x29
	uint32_t volume_id;
	unsigned char volume_label[ 11 ];	//11 bytes
	unsigned char fs_type[ 8 ];		//8 bytes - FAT12
	unsigned char boot_code[ 448 ];		//codigo ejecutable
	unsigned char bootable_signature[ 2 ];	//2bytes-FAT12 0xAA55 si es un sector booteable valido
} NOALIGN stuFsEncabezadoFAT;
```

FS Information Sector (Sector con información del sistema de archivos)

Este sector fue introducido en FAT32 para aumentar la velocidad de acceso de algunas operaciones (en particular, obtener la cantidad de espacio libre). Esta estructura esta ubicada en el sector indicado en el boot record en la posición 0x30 (usualmente el sector 1, inmediatamente después del boot record).
Como dicha estructura no está presente en FAT16, no se encuentra en el sistema de archivos de SODIUM, por lo que tendremos que construirla en su totalidad.

File Allocation Table (FAT - Tabla de asignación de archivos)

La tabla de asignación de archivos (FAT) es una lista de entradas que se asignan a cada cluster en la partición. Cada entrada registra una de las cinco cosas:

• El número de clúster del clúster siguiente en una cadena

• Un carácter especial (EOC) que indica el final de una cadena
• Una entrada especial para indicar un clúster no válido
• Una entrada especial para indicar un clúster reservados
• Un cero para indicar que el grupo no se utiliza

Cada versión del sistema de archivos FAT utiliza un tamaño diferente para las entradas FAT.

FAT12	FAT16	FAT32	Description
0x000	0x0000	0x00000000	Cluster Libre
0x001	0x0001	0x00000001	Valor reservado; no se usa
0x002-0xFEF	0x0002-0xFFEF	0x00000002-0x0FFFFFEF	Cluster usado; el valor apunta al siguiente cluster
0xFF0-0xFF6	0xFFF0-0xFFF6	0x0FFFFFF0-0x0FFFFFF6	Valores reservados; no se usan
0xFF7	0xFFF7	0x0FFFFFF7	Sector defectuoso o cluster reservado
0xFF8-0xFFF	0xFFF8-0xFFFF	0x0FFFFFF8-0x0FFFFFFF	Ultimo cluster del archivo

Tabla 9 – FAT valores de entrada

Para representar estos valores, en SODIUM se utilizan los siguientes defines. Como se observa, tenemos definidos los nombres literales para los valores de FAT12 y FAT 16. Mas adelante, cuando avancemos con la implementacion de FAT32, agregaremos los nombres para dicho sistema de archivos.
Ubicación: **\sodium\include\fs\fat.h**
Codificación:

```
#define FAT12_FREE_CLUSTER          0x000 /* Free Cluster */
#define FAT12_RESERVED_CLUSTER      0x001 /* Reserved Cluster */
#define FAT12_USED_CLUSTER          0x002 /* Used cluster; value points to next cluster */
#define FAT12_BAD_CLUSTER           0xFF7 /* Bad cluster */
#define FAT12_LAST_CLUSTER          0xFF8 /* Last cluster in file */
#define FAT12_ENTRY_SIZE_BITS       12    /* Tamaño de cada entrada en la tabla FAT*/


#define FAT16_FREE_CLUSTER          0x0000 /*Free Cluster*/
#define FAT16_RESERVED_CLUSTER      0x0001 /*Reserved Cluster*/
#define FAT16_USED_CLUSTER          0x0002 /*Used cluster; value points to next cluster*/
#define FAT16_BAD_CLUSTER           0xFFF7 /*Bad cluster*/
#define FAT16_LAST_CLUSTER          0xFFF8 /*Last cluster in file*/
#define FAT16_ENTRY_SIZE_BITS       16     /*Tamaño de cada entrada en la tabla FAT*/
```

Defines para identificar cada tipo de cluster en SODIUM:

```
#define FAT_FREE_CLUSTER        1
#define FAT_RESERVED_CLUSTER    2
#define FAT_USED_CLUSTER        3
#define FAT_BAD_CLUSTER 4
#define FAT_LAST_CLUSTER        5
#define FAT_ENTRY_SIZE_BITS     6
```

El directorio raíz

Este índice es un tipo especial de archivo que almacena las sub-carpetas y archivos que componen cada carpeta. Cada entrada del directorio contiene el nombre del archivo o carpeta (máximo 8 caracteres), su extensión (máximo 3 caracteres), sus atributos (archivo, carpeta, oculto, del sistema, o volumen), la fecha y hora de creación, la dirección del primer cluster donde están los datos, y por último, el tamaño que ocupa.
El directorio raíz ocupa una posición concreta en el sistema de archivos, pero los índices de otras carpetas ocupan la zona de datos como cualquier otro archivo.
Los nombres largos se almacenan ocupando varias entradas en el índice para el mismo archivo o carpeta.
Está presente solamente en FAT16, por lo que en un sistema de archivos FAT32 no debe estar presente.
Igualmente, analizamos esta estructura para FAT16 en SODIUM para comprender el funcionamiento del sistema de archivos actualmente implementado.

Como antes mencionamos, las entradas de directorio tienen atributos. Cuando hablamos de entradas de directorio, estamos incluyendo los archivos. Ahora presentamos los nombres literales que utilizamos en SODIUM para definir los posibles valores que puede tomar el campo 'attributes' de la estructura 'stuFsDirectorioFAT':
Ubicación: **\sodium\include\fs\fat.h**
Codificación:

```
#define FAT_DIRENT_ATTR_RDONLY      0x01    /* Read Only */
#define FAT_DIRENT_ATTR_HIDDEN      0x02    /* Hidden */
#define FAT_DIRENT_ATTR_SYSTEM      0x04    /* System */
#define FAT_DIRENT_ATTR_VOL_LBL     0x08    /* Volume Label */
#define FAT_DIRENT_ATTR_SUBDIR      0x10    /* Subdirectory */
#define FAT_DIRENT_ATTR_ARCHIVE     0x20    /* Archive */
#define FAT_DIRENT_ATTR_DEVICE      0x40    /* Device (internal use only, never found on disk) */
#define FAT_DIRENT_ATTR_UNUSED      0x80    /* Unused */
```

Entrada en un directorio FAT

Esta estructura identifica cada entrada de directorio. Como podemos apreciar es similar tanto para un archivo como para un directorio. Se diferencian por el campo 'atributes' como mencionamos anteriormente. Analizando dicho campo, en SODIUM sabremos la forma de tratar dicha entra de directorio.
Ubicación: **\sodium\include\fs\fat.h**

```
typedef struct stuFsDirectorioFAT {
        char filename[8];
        char extension[3];
        uint8_t attributes;
        uint8_t reserved;
        uint8_t create_time_fine;
        uint16_t create_time;
        uint16_t create_date;
        uint16_t access_date;
        uint16_t ea_index;
        uint16_t modified_time;
        uint16_t modified_date;
        uint16_t starting_cluster; //Cluster de comienzo del archivo relativo al final del ultimo sector de ROOT
        uint32_t filesize;
}NOALIGN stuFsDirectorioFAT;
```

El atributo 'ea_index' (es usado por OS/2 y NT) en FAT12 y FAT16, hace referencia a los 'extended object attribute (EA)'. Queda pendiente el análisis del SODIUM para determinar si se utiliza este campo o no. En FAT32 representa los dos primeros bytes del primer cluster del archivo, dicho número de cluster se completa con el campo 'starting_cluster' interpretado como los dos últimos bytes del primer cluster de dicho archivo. El resto de la estructura es similar tanto para FAT16 como para FAT32.

A continuación presentamos los posibles valores especiales que puede tomar el 1er byte del campo 'filename' en la estructura 'stuFsDirectorioFAT'.
Cualquier otro valor, significa el valor del primer caracter del nombre de archivo.
Ubicación: **\sodium\include\fs\fat.h**

```
#define FAT_EMPTY       0       // Campo aun no usado (luego del formateo)
#define FAT_SUBDIR      0x20    // El archivo es un directorio
#define FAT_DELETED     0xE5    // El archivo fue borrado
```

Otras definiciones útiles en Sodium respecto a Fat16:

Ubicación: **\sodium\include\fs\fat.h**

Magic number que tienen las FAT (para cada versión):

```
#define FAT12_HEADER        "\xf8\xff\xff"
#define FAT12B_HEADER       "\xf0\xff\xff"
#define FAT16_HEADER        "\xf8\xff\xff\xff"
```

Size del header:

```
#define FAT12_HEADER_SIZE 3
#define FAT16_HEADER_SIZE 4
```

PARTE 2: Investigación sobre la implementación del sistema de archivos en SODIUM

Nuestro primer acercamiento al sistema de archivos del SODIUM tiene como principales objetivos los siguientes:

1. Crear el Handler para FAT16/32 (Es una Estructura)
2. Reconocer una FAT16/32
3. Leer un Archivo en disco FAT16/32

Descubrimos que para realizar esto, era necesario leer y entender el funcionamiento de la función (y los método que llama) "ifnBuscarArchivoFat(*char, &Struct, HandleFat12)".
Busca un archivo de nombre determinado en el filesystem apuntado y devuelve la entrada encontrada en el buffer recibido por parametro.

Ubicación:
Definición: **\sodium\include\fs\fat.h**
Codificación: **\sodium\fs\fat.c**

Para lograr un mejor entendimiento, realizamos los siguiente:

1. En primera instancia levantamos el sodium y leímos la pseudo ayuda que ofrece.
2. Luego ejecutamos el comando "leer" que ofrece el shell con un archivo llamado "lote1.txt"
3. Vimos que efectivamente el comando muestra el contenido del archivo por pantalla con unas cabeceras que muestran cierta información de cual es el cluster de inicio y cuantos sectores ocupa.
4. modificamos el makefile del sodium para que incorpore un archivo mas (lote2.txt) definidos por nosotros conteniendo un texto cualquiera ("hola mundo!" para variar) y lo listamos desde el sodium mismo para ver si la info de cabecera cambiaba.
5. por la info que mostraba vimos que las funciones dentro del sodium son llamadas en la siguiente secuencia:

- Shell.c :::: vfnMenuLeeArch(*char nombre)
- Fat.c ::::::: vfnMostrarArchivo(HandleFat12, *char nombre)
- Fat.c ::::::: ifnBuscarArchivoFat(*char nombre, &Struct, HandleFat12)
- Fat.c ::::::: vfnImprimir(&Struct)
- Fat.c ::::::: ifnLeerArchivoFat(*char nombre, &Struct, buffer)

Con respecto al Handle, tiene esta forma:

```
stuFsHandleFat
{
        Dispositivo
        Encabezado Fat 12
        Handle para Directorios Fat12
        Bits de Estado
}
```

Ubicación: **¥sodium¥include¥fs¥fat.h**

```
enum EstadosHandleFat
{
        NoInicializado=0,
        InicializadoOK=1,
        InicializadoNOK=2,
};
```

```
typedef struct stuFsHandleFat
{
        struct stuDeviceHandle *device;
        struct stuFsEncabezadoFAT *header;

        uint8_t *fat1;
        uint8_t *fat2;
        uint8_t *fat3;

        struct stuFsDirectorioFAT* root_dir;

        /* funciones paraleer_header() adaptarse dinamicamente a FAT12/16: */
        pFatXConst get_constant;
        pFatXGetNextCluster get_next_cluster;
        pFatXSetNextCluster set_next_cluster;
        enum EstadosHandleFat estado_handle_fat;
} stuFsHandleFat;
```

Como primer objetivo nos proponemos implementar un LeerFat32 u/o LeerFat16 que lea del primer disco rígido un archivo como ya lo hace en el diskette. Solo siguiendo el mismo esquema que tiene hoy el sodium para leer, pero utilizando las interrupciones a disco, pero para esto tenemos que armar un handle que pueda transportar la información como lo hace el actual handle de fat 12.

Para mas adelante dejaríamos la parte de reconocer FAT16 o FAT32 durante el booteo. Una vez implementada la parte de lectura desde el primer disco, se buscará que bootee desde el mismo disco.

Una posibilidad para lograr dicho booteo sería desde el bochs hacer con mkfs una partición fat16/32 en un archivo imagen, como se hace hoy para fat12, y modificar los parámetros que se le pasan al bochs para que además de bootear desde ese diskette ficticio, anexe esa imagen fat 16/32 como un disco en el canal IDE0 como master. Nos basaríamos en la documentación del bochs para facilitarnos la tarea.

Dejamos por último una aclaración de uno de los docentes de la cátedra que deberá ser tenida en cuenta mas adelante en la implementación: "unos de los problemas con los que se van a encontrar si quieren trabajar con FAT32 es que, actualmente el sodium trabaja en modo real, esto hace que se deban utilizar obligatoriamente los registros de 16 bits de coprocesador y no los de 32. Por que en caso de utilizar los registros de 32 correríamos el riesgo de para tablas FAT muy grandes, no dar a basto en la memoria (con tablas mayores a 1MB)".
Con lo cual llegaba a la conciliación que íbamos a tener que trabajar por partes un registro de 32 bits traído (o llevado) al micro.

Conclusión

Analizar las estructuras actuales del sistema de archivos del SODIUM nos sirvió para entender como se almacena la información en dicho sistema, y para reconocer dichas estructuras, definiciones y valores caracteristicos de FAT 16 en nuestro sistema operativo.
Estos conocimientos fueron los cimientos necesarios para poder comenzar a investigar el modo en que se llega a cabo la administración de dicho sistema de archivos. Una vez terminada dicha investigación definiremos las modificaciones o agregados necesarios para poder implementar el sistema de archivos FAT32 en SODIUM.

Referencias

[1] http://www.microsoft.com/hwdev/download/hardware/fatgen103.doc
[2] Paper FAT.Diferencias.Entre.Fat16.Y.Fat32.doc
[3] Código fuente del SODIUM

Datos de Contacto:

Aldo Flores | aldoh84@gmail.com | Universidad Nacional de La Matanza

Pablo Sasson | pablo_sasson@hotmail.com | Universidad Nacional de La Matanza
Pablo Groba | pgroba@gmail.com | Universidad Nacional de La Matanza
Claudio D'Amico | nfs5cgd@gmail.com | Universidad Nacional de La Matanza

Diseño

Cuando finalizamos la investigación sobre el sistema de archivos FAT de Microsoft y terminamos de interpretar como estaba implementado en el SODIUM, el siguiente paso fue completar el modulo de dicho sistema de archivos en nuestro sistema operativo. Cuando hablamos de completar, hablamos de implementar FAT16 y FAT32 en SODIUM, porque ya estaba implementado FAT12 en su totalidad.

Pero a la hora de avanzar con esta tarea, nos encontramos con una serie de complicaciones que fueron cambiando nuestra investigación de forma que se agregaron nuevas etapas al proceso de implementación del sistema de archivos en el SODIUM.

Habiendo agregado etapas no previstas al inicio de la investigación, nos vimos obligados a redefinir nuestros objetivos. A principio nuestra meta era implementar por completo el modulo FAT en Sodium. La investigación fue tomando distintos rumbos. Finalmente, nuestro trabajo consistió en crear un ambiente de pruebas adecuado para nuestro trabajo y también que facilite la tarea de investigaciones futuras. Luego lograr el booteo del sistema operativo tanto en una partición formateada en FAT16 como en una formateada en FAT32. Además, implementar un driver básico de disco que permita leer sectores del mismo. Y por ultimo, dejar documentación sólida para investigaciones futuras tanto del driver de disco (para poder ser completado), como para el sistema de archivos FAT16 y FAT32.

A continuación vamos a hacer mención a cada una de estos inconvenientes, y explicar como nos afectaron e hicieron que modifiquemos el rumbo de nuestra tarea.

El punto más importante y decisivo, fue la fuerte dependencia que presentaba el sistema de archivos del dispositivo físico. Poder resolver este problema, no solo fue prioridad a la hora de diseñar nuestro trabajo y pensar soluciones para los inconvenientes encontrados, sino que tiene que ser la base de futuras investigaciones relacionadas con drivers y sistemas de archivos en lo que al SODIUM se refiere.
Para explicar mejor este tema podemos mencionar que cuando se intentaba leer o escribir en FAT12, las funciones del sistema de archivos llamaban a funciones del driver de disquete explícitamente. Es decir, como estaba estructurado el sistema de archivos, solo se podría leer FAT12 en un disquete; ni en un disco rígido ni en un dispositivo USB. Por lo tanto, para poder manejar FAT12 en un disco rígido por ejemplo, deberíamos reescribir todas las funciones para que llamen a las funciones del driver IDE.
Lo que nosotros planteamos como solución, es una implementación conocida como LATE-BINDING. Esto es, enlazar las funciones de inicialización, lectura y escritura de los sistemas de archivos en tiempo de ejecución a las funciones de inicialización, lectura y escritura de los dispositivos físicos. Dependiendo del dispositivo con el que estemos trabajando, el sistema de archivos utilizará las funciones del driver correspondiente.
En las estructuras que se utilizan en el sistema de archivos, se utilizan punteros a funciones, que apuntarán a funciones de diferentes drivers en tiempo de ejecución, automatizando el proceso de lectura y escritura, y logrando independizar al sistema de archivos del driver de dispositivo.

Una vez que tuvimos una buena definición sobre como implementar la relación entre el sistema de archivos, nos encontramos con una nueva complicación: la ausencia de un driver de disco con el cual probar FAT16 y FAT32 en algún dispositivo de almacenamiento masivo. Por lo tanto, el siguiente paso era obtener un driver de disco IDE-ATA con las distintas formas de acceso: utilizando el bus PCI, utilizando PIO y utilizando DMA.

Una vez elegido el driver de disco que mas se adaptaba a nuestras necesidades, nos encontramos con la ausencia de un ambiente de pruebas adecuado para realizar implementación del driver ATA.

Cuando hablamos de ambiente de pruebas, nos referimos a poder tener un disco virtual totalmente configurable al cual acceder para leer y escribir datos en él. Y también desde el cual poder iniciar la maquina virtual, sin la necesidad de iniciarla desde un disquete virtual como se hacia hasta entonces.
Por lo cual, nuestro primer paso de la investigación fue generar un ambiente de pruebas adecuado; configurable fácilmente en tiempo de compilación, que nos dé la posibilidad tanto de contar con un disco virtual formateado en distintos sistemas de archivos, como así también poder separar el proceso de compilación del sistema operativo en diferentes formas de booteo virtual.
Estas diferentes formas de booteo virtual serían: iniciar en un disquete formateado en FAT12, iniciar en un disco rígido formateado en FAT16 por ultimo, iniciar en un disco rígido formateado en FAT32.

Teniendo funcionando el ambiente de pruebas y, por ende, pudiendo iniciar la maquina virtual con el disco rígido virtual creado en tiempo de compilación, nuestro siguiente paso en la investigación fue lograr iniciar el sistema operativo instalado en una partición de disco rígido formateada tanto en FAT16 como en FAT32.

Habiendo logrado satisfactoriamente este paso, pusimos en practica un driver de disco elemental que inicia los dispositivos IDE existentes en la maquina, muestra la geometría y datos del disco rígido por pantalla, realiza la lectura de un sector de disco y muestra por pantalla el resultado de la lectura; es decir, si la misma es satisfactoria o no.

Como esquema definitivo en el sistema operativo SODIUM, proponemos implementar un modulo FATX. Es decir, un solo modulo para los diferentes sistemas de archivos FAT (12, 16 y 32) que utilice diferentes funciones dependiendo del sistema de archivo del dispositivo que se esta actualizando en ese momento.
Y por otro lado tener drivers para los distintos dispositivos de almacenamiento. Es decir, disquete, disco rígido y dispositivos USB.
En tiempo de ejecución se haría el enlace de las funciones como se explicó anteriormente para lograr una independencia total del dispositivo de almacenamiento.

Esquema Final:
Modulo FATX ---> Modulo FDD
---> Modulo ATA (No existente)
---> Modulo USB (No existente)

Implementación

Ambiente de prueba
Si de desarrollar un modulo para manejar cualquier FileSystem se trata, es necesario contar con un dispositivo físico donde ejecutar las pruebas, o bien, un dispositivo virtual para tal fin. Sabíamos que el proyecto ya contaba con un emulador real de hardware, como es el Bochs, que nos permitiría en un principio poder emular un disco y así no tener que, por cada compilación, copiar a un disco real todo el SO. El problema radicaba en que esta parte aun no estaba terminada en el ambiente de prueba que ya proporcionaba sodium.

Cabe aclarar que el responsable de levantar en ambiente de prueba dentro del proyecto es un "Makefile" que se encuentra dentro de la carpeta "tests" dentro de la raíz del proyecto. Este contiene todas las órdenes necesarias para compilar todo el proyecto, hacer una imagen de diskette, configurar el bochs y finalmente, levantarlo. En cuanto a la imagen de disco, estaba anulada por considerarse inestable. La intervención nuestra lo que hizo fue utilizar una serie de comandos extras para estabilizar la creación del disco virtual. Estos comando son:

Bximage: Utilizado para reservar el espacio dentro del disco real. Lo que hace es generar un archivo de acuerdo con el tamaño de disco que se desea trabajar y generar los archivos de configuración para que el bochs lo detecte. El contenido de este archivo son todos caracteres "null" ó "0x00".

Parted: Utilizado para establecer la firma base del disco, en nuestro caso "msdos"

sfdisk: Utilizado para crear las particiones físicas del disco. Es necesario utilizar este comando por que permite redirección de comandos por medio de pipes.

grep/tail/head/cut/sed: Utilizados para calcular el offset a particiones, en nuestros ejemplos solo se utilizan 2 particiones, pero se pueden utilizar mas y recalcular dinámicamente el offset a través de la concatenación de estos comando agregando una nueva linea de ellos.

Bc: es el responsable de ejecutar la operación matemática de multiplicación, necesaria para saber la cantidad de sectores

losetup: Utilizado para poder asociar a un dispositivo real del sistema operativo anfitrión el disco virtual para poder montarlo y copiar los archivos del sodium. Esta asociación tiene en cuenta que hay que efectuarla al offset de inicio de partición dentro del archivo (nuestro disco virtual en este caso) por que sino el comando mount no reconocería la misma como valida.

dd: Utilizado para copiar la cabecera de la partición, en este caso el pbr.bin o pbr32.bin, que contiene el bootstrap del sodium.

mount/umount/cp: Utilizado para montar la partición y copiarle los archivos que necesita el sodium para iniciar, los mismo serán recorridos desde la cadena de clusters y llevados a memoria desde el bootstrap y el loader.

Una vez obtenida una creación de disco virtual estable, se decidió desdoblar la compilación bajo el entorno de prueba en dos. Esta se ejecutaba bajo la orden "make bochs_c". Como era necesario probar, tanto el Fat16 como en Fat32, que booteaba desde disco se remplazo la vieja orden antes mencionada por las siguientes dos:

make bochs_c16: Crea un disco de 1Gb con 1 particion fat16, el tamaño de disco puede ser regulado por la variable SODIUM_H16_IMG_SIZE.

make bochs_c32: Crea un disco de 400Mb con 2 particiones (fat32 {100mb}, ext2 {300 mb}), el tamaño de disco puede ser regulado por la variable SODIUM_H32_IMG_SIZE.

La cantidad de particiones se regulan desde el trabajo en conjunto que proporcionan los comandos echo y sfdisk (lineas 224 y 258 respectiva mente para cada compilación). Para mas detalles de como editar la cantidad de particiones leer la documentación del comando sfdisk.

Seria interesante probar, en un esquema multiparticion que ya ofrese este entono de pruebas, copiar el sodium en una partición que no se la inicial del disco, y bootear desde ahí. Es razonable para nosotros aclarar esto dado a que el sodium se encontrara mayormente en este escenario en instalaciones reales. Para esto es necesario que, al formatear las particiones el valor de BPB_HiddSec[1] este lleno con el verdadero offset de la partición. Se hablo con el equipo de desarrollo del BOOM para que al bootear se pasara como dato en el DI este offset de partición.
Cabe destacar que la modificación del entorno de pruebas y la inclusión de los archivos pertenecientes a las dos etapas de booteo, tanto en Fat16 como en FAT32, no se remiten únicamente al makefile de la carpeta "tests" del proyecto, sino también al de la carpeta "solo" donde se incluyeron las ordenes necesarias para obtener correctamente los archivos binarios pbr.bin, pbr32.bin, loadhdd.sys y loadh32.sys.

Concluiremos que este ambiente de pruebas resultara muy util para todas las pruebas que se hagan que impliquen al menos un acceso a disco (formateo, gestión de particiones, hibernación, y funciones básicas de un filesystem) en tanto no este solida la parte de instalación propia del sistema operativo.

El booteo del Sodium
Normalmente los equipos con capacidad de booteo desde disco rígido detectan la partición activa dentro de los mismos y copian los primeros 512 bytes de dicha partición en la dirección de memoria 0x7C00 si es que tienen la firma de booteo 0x55AA. Una vez hecho esto se le cede la ejecución a dicho código. En efecto el Sodium no escapa de este comportamiento. Es aquí donde se ejecutan las primeras instrucciones del SO, que estarán destinadas a iniciar el mismo.

Esto ocurre en dos etapas, la primera, encargada solo de llevar a memoria el loader.sys y cederle el

control al mismo. La segunda, recorrer un listado de archivos definidos en tiempo de compilación y cargarlos en memoria en las posiciones definidas en el mismo archivo. Tanto para una etapa, como para la otra, es necesario poder recorrer la FAT y el RootDirectory para obtener los archivos necesarios para avanzar a la siguiente etapa. El código de ambas etapas se puede encontrar dentro de la carpeta "solo" del proyecto bajo los nombres de bootloader.asm (Etapa 1) y loader.asm (Etapa 2). El problema con el que nos encontramos al principio, entorno a hacer que bootee el So desde disco, es que todo este código, tanto el de la primer como el de la segunda etapa, estaba preparado para funcionar solo en FAT12, es decir, en diskette.

Ambos códigos, para poder recorrer la cadena de cluster que conforman cada archivo que es necesario llevar a memoria, cargaban en memoria TODA la FAT, lo cual para un diskette de 1.44 Mbyte no representa un gran desperdicio de espacio. Esto resulto una complicación a la hora de desarrollar los nuevos bootloaders y loaders para FAT16/32 ya que no resulta un alto desperdicio de memoria una tabla FAT completa que pueda direccionar 1.44 Mbytes (4.5 kb) dentro del espacio de direccionamiento posible en modo real que es de 1MByte. Pero a la hora de bootear desde una partición mayor a 400 Mbytes es inviable tomar esto como una opción, dado el poco espacio con el que se cuenta en modo real.

Tal es así que, en un principio, se decidió por temas de práctica, seguir con el esquema que se planteaba en los dos niveles de inicio de diskette y se intento cargar toda la FAT para su versión de 16 bits incluyendo su copia. Esto representó, tomando clusters de 8KBytes, mas de 100 Kbytes por FAT, lo cual tendríamos 200 Kbytes, la quinta parte del espacio de direccionamiento total, de los cuales solo se utilizaban los primeros 40 bytes para direccionar todos los archivos necesarios para bootear el Sodium.

Esto trajo aparejado también la adaptación de todo el código para soportar cambios de segmento. Teniendo en cuenta que solo puedo direccionar de 64 Kbytes, para llegar al RootDirectory que se ubicaba a continuación de las dos copias de FAT se deberían hacer más de cinco (5) cambios de segmentos.

Fue por esto mismo que se decidió solo copiar los primeros 2 Kbytes de UNA sola de las FAT's, tanto en su versión de 16 bit's (FAT16) como en su versión de 32 bit's (FAT32).

Concluiremos que para el desarrollo de las dos etapas para booteo en FAT16 y las otras dos correspondientes a FAT32 se utilizaron como base los archivos fuente ya existentes para diskette, pero con varias reformas. Dos archivos independientes para FAT16 y otros dos independientes para FAT32. Ellos se detallan a continuación:

FAT16
- pbr.asm
- loadhdd.asm

FAT32
- pbr32.asm
- loadh32.asm

Todos ellos se encuentran dentro de la carpeta "solo" del proyecto. Si se desea profundizar en las modificaciones, dentro de la documentación de proyecto se proveen los diff's y changelog's donde se ven mas ampliados estos cambios.

En cuanto a como queda la disposición de archivos en memoria concluidas las dos etapas, seria así:

```
+--------+
|        |
|LISTADO | 8000:0000 tam:266 bytes
|SHUTDOWN| 7280:0000 tam:12288 bytes
|PS.BIN  | 6F80:0000 tam:12288 bytes
|TST_FLT | 6C60:0000 tam:12800 bytes
|TST_IPC | 6920:0000 tam:13312 bytes
|USUARIO | 6620:0000 tam:12288 bytes
|SHELLUSR| 5F00:0000 tam:29184 bytes
|INIT.BIN| 5BE0:0000 tam:12800 bytes
|IDLE.BIN| 5BC0:0000 tam:512 bytes
|MBR.BIN?| 5BA0:0000 tam:512 bytes innesesario
|KEYMAPS | 5AC0:0000 tam:3584 bytes
|PBR.BIN?| 5AA0:0000 tam:512 bytes innesesario
|--------|
|        |
|LISTADO |
|        |
|        |
|--------| 27E0:1100
|        |
| Sector |
| Datos  |
|--------| 27E0:0B00
|        |
| FAT32  |
|        |
|--------| 27E0:0500
|        |
|LOADER  |
|        |
|--------| 27E0:0000
|        |
|SODIUM. |
|SYS     |
|        |
|--------| 07E0:0000 tam:338944 bytes
|        |
+--------+
```

El problema de este esquema es que, en cuanto el sodium crezca en funcionalidad y tamaño, muy probablemente pise el loader, ocasionando una interrupción en la carga. Por ende se recomienda que el inicio del loader en la primera etapa este por encima del segmento 9000:0000, teniendo cuidado de solo copiar parte de la FAT, ya que mas allá del segmento A000:0000, se encuentra la zona reservada por el bochs para la emulación de la VM.

El modulo IDE[5]

Si bien logramos terminar con la creación del ambiente de prueba y el booteo tanto en FAT16 como en FAT32 casi finalizando el año, nos propusimos dejar un driver IDE básico funcionando. Se lo califica como básico por la amplia diversidad de controladoras IDE del mercado, solo se codifican las funciones básicas sin contemplar los "features" particulares de cada una de ellas. Esto permitirá que cuando se continúe con ésta investigación, sea más fácil completar dicho driver.

Un driver IDE tiene diferentes forma de transmitir la información entre dispositivo y memoria: PIO, DMA. Estos dos pueden ser a través del puente puente PCI o desde el puente ISA básico que provee cualquier placa base. Nosotros completamos la inicialización, lectura y escritura en modo PIO (esto es, utilizando puertos y buffers de memoria) a través del puente PCI. Esto se hace pasándole a la función ide_initialize los puertos base donde se encuentra generalmente las controladores IDE (PATA). La idea seria que en un futuro, cuando este implementado el modulo PCI, este sea el encargado de pasarle a la función ide_initialize la dirección base de la controladora IDE que detecto, sea PATA o SATA. Este modulo ya se escapaba de nuestras manos dado a la complejidad y diversidad que se presenta en el mercado de diferentes controladoras PCI. En resumen, una amplia biblioteca IDE combinada con una amplia biblioteca PCI harían la combinación perfecta de features aprovechables para cada entorno en el que el SO corra. Se ha dejado abierta la posibilidad de introducir las funciones de lectura en modo DMA dentro de este modulo (fn: ide_ata_access() ; linea 780)

A modo de prueba se creó el comando de shell 'leerdisco'. Si se ejecuta este comando una vez inicializado el SODIUM, inicializa el dispositivo, trae datos del disco e intenta leer un sector informando por pantalla el resultado de la operación.

Cabe destacar que no nos fue posible probar exhaustivamente el driver, pero con la investigación que nosotros realizamos y el código presente en SODIUM actualmente, no debería ser difícil mejorar las funciones presentes. Además, en la presente documentación del proyecto que es acompañado por este informe dejamos a disposición de la cátedra varios ejemplos de drivers que facilitarán la tarea a los alumnos que tengan que continuar nuestra investigación.

Archivos del driver:

4. kernel/drivers/Ide.c
5. include/kernel/drivers/Ide.h

Ambos archivos se encuentran exhaustivamente documentados, describiendo paso a paso y en términos técnicos lo que el modulo hace, lo cual no le sera complicado a la siguiente generación de investigadores seguir esta secuencia.

Como extra, dado que la base de este código ofrece la posibilidad de hacer lecturas ATAPI (cdrom) se deja un driver modelo[6] para la codificación de dicho modulo. También se deja un ejemplo para soportar montajes de otros filesystem[7] (para el futuro comando mount).

El modulo FAT

El modulo fat que encontramos en sodium a principio de año, contaba con todo lo necesario para el funcionamiento en fat12 y tenia parte de fat16 ya armada, pero no estaba concluida.

Dentro del modulo (fs/fat.c) encontramos estructuras como la del Encabezado, que contienen datos del primer sector del disco y que describen un file system. Luego otra estructura que describe las entradas al directorio raíz. Esta solo se utiliza para fat12 y en el futuro para fat16, no así para fat32, que no cuenta con el Área de Directorio. Debemos aclarar que si bien cambia la ubicación del Directorio Raíz en FAT32 (Se encuentra dentro del sector de datos y no en un área aparte y delimitada como en FAT16 y FAT12), dicha estructura sigue siendo de utilidad también en este sistema de archivos. Luego cuenta con una estructura que es el handler, que posee punteros a las estructuras recién nombradas, a las copias de las fat's propiamente dichas y a las funciones dinámicas que setean y obtienen el próximo cluster a leer. Para poder manejar dinámicamente fat12 y 16, se aisla el código dependiente de cada uno en funciones separadas, y al handler se le carga punteros a esas funciones (late-binding).

Como se menciono al principio, este modulo tiene un alto nivel de acoplamiento con el modulo de acceso a diskette en sus funciones de lectura y escritura, quizás por que en un principio, solo se pretendía que el SO funcione en este medio. Dadas estas condiciones, y todas las restricciones con las que nos encontramos nos fue imposible hacer grandes avances en este área, pero proporcionamos una documentación solida de las diferencias entre ambos filesystems[4] y el estudio de un posible modulo candidato[3] que reemplace el actual modulo. Este ultimo provee la abstracción necesaria del dispositivo de bloque necesario para la lectura y escritura bajo la especificación del filesystem[1] oficial a través del concepto de latebinding. Esto permitiría que el modulo sea llamado indistintamente de cualquier dispositivo que se monte una vez que el SO se halla cargado, esto seria, primero detectar el dispositivo de bloques, inicializarlo, luego asociar las funciones propias del dispositivo al handler del modulo, inicializar una instancia del modulo y finalmente hacer la funciones de lectura/escritura bajo el filesystem detectado (FAT12/16/32). Esto se puede hallar en la carpeta que acompaña esta documentación llamada: "Futura Implementación FAT (Driver FAT)"

Referencias

[1] http://www.microsoft.com/hwdev/download/hardware/fatgen103.doc
[2] http://wiki.osdev.org/IDE
[3] fat_io_library.zip extraido de http://www.pjrc.com/tech/8051/ide/fat32.html
[4] FAT.Diferencias.Entre.Fat16.Y.Fat32.doc
[5] Programming Interface for Bus Master IDE Controller Revision 1.0 5/16/94 – Esta es la documentacion base de que todos los fabricantes de controladores IDE usan. Se puede encontrara dentro de la Docuemntacion que acompaña este informe como idems100.pdf.
[6] iso9660.c
[7] filesystemAlter.c

Alcance

En el transcurso de este año los avances fueron grandes. Además del lograr objetivos que van a servir en el desarrollo del sistema operativo en el futuro, pudimos contribuir en la investigación de otros grupos este mismo año. Como ser, booteo en dispositivos USB y aportamos conocimientos teóricos al grupo que desarrollo el multibootloader de Sodium. Lo más importante fue haber logrado lo siguiente:

1. **El arranque en FAT16 y 32:** Para esto tomamos cada una de las partes que forman parte del arranque de Sodium, y le hicimos todas las modificaciones necesarias para poder logar los ya mencionados tipos de arranque. Creamos dos archivos para FAT16 y otros dos para FAT32, es decir, los arranques son independientes uno de otro.

   FAT16
   pbr.asm
   loadhdd.asm

   FAT32
   - pbr32.asm
   - loadh32.asm

2. **Poder generar un buen ambiente de prueba:** Logramos la generación de discos virtuales para poder probar en sus particiones los distintos tipos de arranque.

3. **Y por ultimo tener el "puntapié inicial" con respecto al driver:** A modo de prueba se creó el

comando de shell 'leerdisco'. Si se ejecuta este comando en la ventana de shell del SODIUM, inicializa el dispositivo, trae datos del disco e intenta leer un sector informando por pantalla el resultado de la operación

El día de mañana cuando la investigación continúe, lo que se tendría que lograr es lo siguiente:

6. A nivel modulo FAT: lograr la independencia con el dispositivo; es decir, separar lo que actualmente esta fuertemente unido: las funciones de acceso con las funciones de floppy, para luego lograr un handler único donde las funciones de acceso sean punteros a las funciones particulares de cada dispositivo (latebinding). Para esto nosotros habíamos encontrado una biblioteca de FAT que al parecer es muy completa y modular, que va a estar en el CD que entregamos junto con la última versión de Sodium.

7. Desde el punto de vista del driver: hoy por hoy esta funcionando la inicialización de los dispositivos, y las funciones de read y write a disco a través de PIO. Lo ideal seria seguir desarrollando el driver hasta lograr, entre otras cosas, el trabajo con DMA y poder utilizar el canal PCI.

El motivo por el cual estos objetivos no pudieron alcanzarse fue porque no teníamos la base necesaria para lograrlos. Es decir, si uno analiza bien lo que queda hacer el día de mañana, es básicamente lo que nosotros teníamos pensado desarrollar. Pero para eso, nos era indispensable tener la posibilidad de cargar en memoria la FAT por completo, poder generar un buen ambiente de prueba y tener el IDE desarrollado. Con los objetivos alcanzados a lo largo del año, nosotros dejamos para los grupos venideros la base para que se pueda lograr el objetivo de tener un modulo FAT independiente del dispositivo, y mejorar el driver para el acceso a disco.

Bibliografía

[1] http://www.microsoft.com/hwdev/download/hardware/fatgen103.doc
[2] http://wiki.osdev.org/IDE
[3] fat_io_library.zip extraido de http://www.pjrc.com/tech/8051/ide/fat32.html
[4] FAT.Diferencias.Entre.Fat16.Y.Fat32.doc
[5] Programming Interface for Bus Master IDE Controller Revision 1.0 5/16/94 – Esta es la documentacion base de que todos los fabricantes de controladores IDE usan. Se puede encontrara dentro de la Docuemntacion que acompaña este informe como idems100.pdf.