Christian García Mascarell | Josep Jordana Mitjans

christian.garcia@students.salle.url.edu | josep.jordana@students.salle.url.edu

Infinity War

Práctica SOA

Índice

[1. Introducción 2](#_Toc7015427)

[1.1. FAT32 2](#_Toc7015428)

[1.2. EXT4 4](#_Toc7015429)

[2. Investigación 7](#_Toc7015430)

[2.1. Detección del sistema de ficheros 7](#_Toc7015431)

[2.1.1 FATX y EXTX 7](#_Toc7015432)

[2.1.2 FATX 8](#_Toc7015433)

[2.1.3 EXTX 9](#_Toc7015434)

[2.2. Extracción de la meta-data 10](#_Toc7015435)

[2.2.1. FAT32 10](#_Toc7015436)

[2.2.2. EXT4 14](#_Toc7015437)

[2.3. Localización de un fichero y meta-data 18](#_Toc7015438)

[2.3.1. FAT32 18](#_Toc7015439)

[2.3.2. Ext4 19](#_Toc7015440)

[2.4. Obtención de los datos de un fichero 23](#_Toc7015441)

[2.4.1. FAT32 23](#_Toc7015442)

[2.4.2. Ext4 23](#_Toc7015443)

[3. Estructuración de la práctica 24](#_Toc7015444)

[3.1 Módulos y funciones que se usarán 24](#_Toc7015445)

# Introducción

Este documento tratará de explicar los distintos sistemas de ficheros que se desean analizar (FAT32 y EXT4) y en cómo se deberán de analizar los sistemas de ficheros con tal de obtener información sobre los mismos.

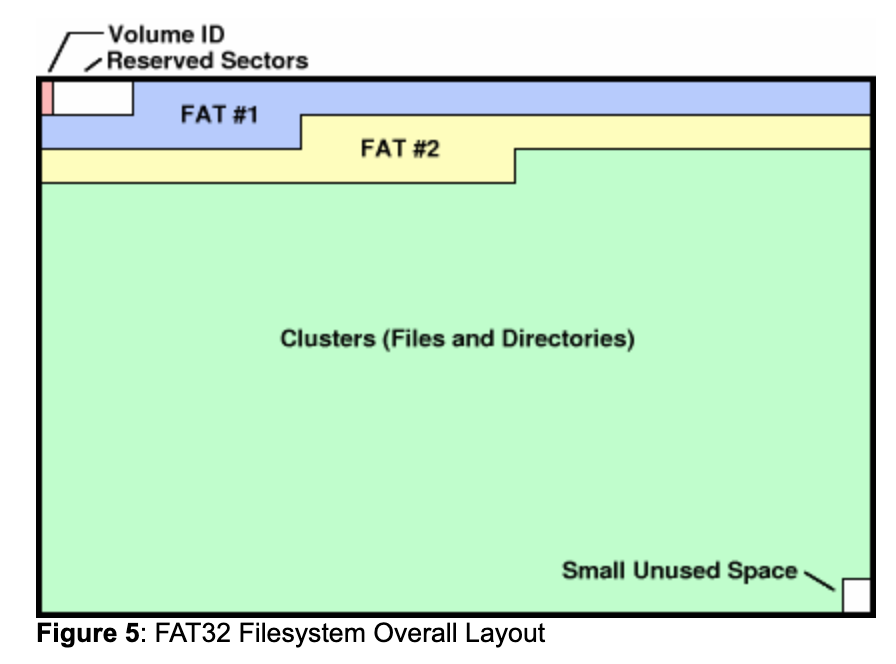
## FAT32

FAT32 es un sistema de sistema de ficheros introducido el 1996 por Microsoft. Este, junto con FAT12, FAT16 y otros (como EXFAT), forma parte de la familia de sistemas de ficheros FAT, categorizados de esta manera por el uso de una *File Allocation Table*, que no es nada más que una estructura que indica, para un fichero (un directorio también es un fichero), donde se encuentra su información en el disco.

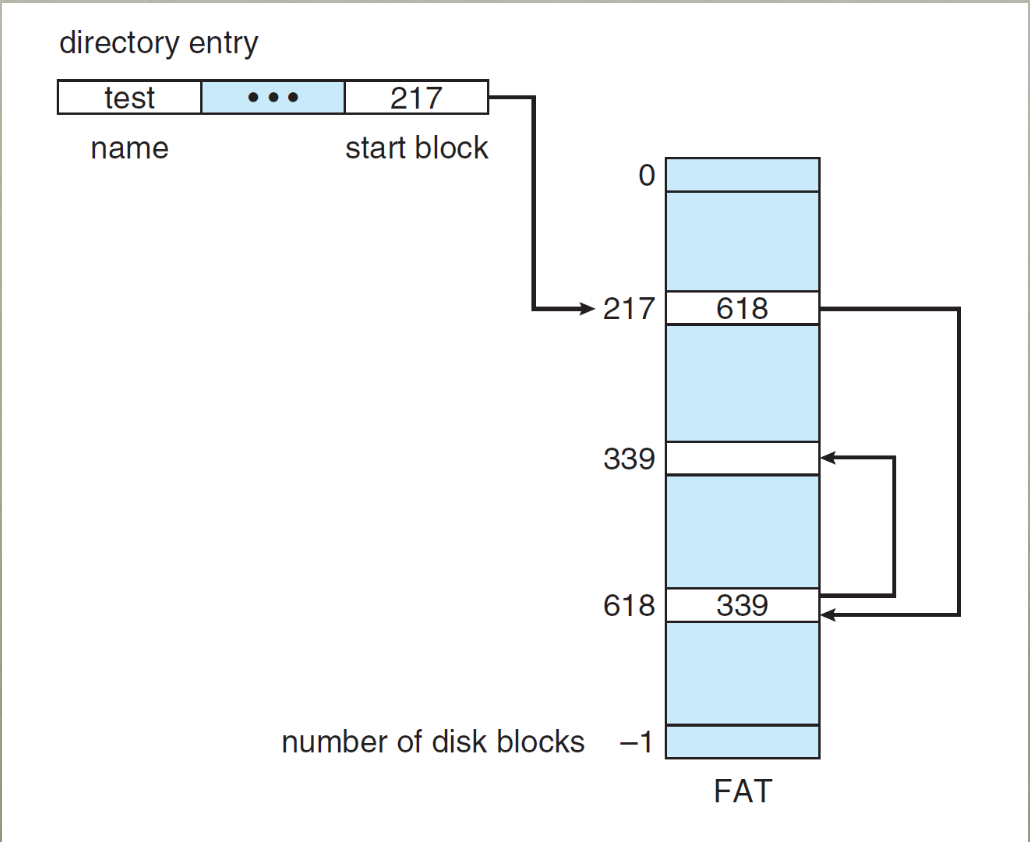
Dicho esto, esta definición es aplicable a muchas estructuras de otros sistemas de ficheros y solo marca para qué sirve, no cómo lo consigue, que es lo que hace una u otra estructura a la vez diferente y mejor o peor que las otras. En el caso de la tabla FAT en FAT32, se puede decir simplemente que esta es un array de valores de 32 bits.

Cada valor de los 32 bits (los 4 superiores están reservados) representa un clúster (2^28 clústeres como máximo), la unidad de datos más pequeña de FAT32 (un clúster puede ser 1, 2, 4, 8, 16, 32,64 o 128 sectores de 512, 1024, 2048 o 4096 bytes).

Cada clúster tiene asociado a la vez un espacio en el espacio de data (clústeres) y una posición en la tabla, el valor de la cual, en el segundo caso, es el siguiente clúster del fichero, y así sucesivamente hasta que se llegue al final de este.



Por lo tanto, la recuperación de la información del fichero se hace en un formato *Linked List*, tal que el directorio que contiene el fichero (que puede ser otro directorio) no tiene toda la información para acceder a todos los datos de este, solo el número del primer clúster.



Esta forma de estructurar los datos le da ventajas importantes, entre las cuales está el hecho de que no hay fragmentación externa, solo interna (mucho menos importante), y que un fichero puede crecer fácilmente.

Por otro lado, también están las desventajas, entre las cuales se encuentra un tiempo de acceso aleatorio muy malo (se debe recorrer todo el espacio) y un espacio de meta-data muy grande (la FAT ocupa bastante, y más si hay *backups*).

Un dato muy importante que mencionar es que todo el sistema de ficheros FAT se encuentra en *Little Endian* (el byte más significativo es el último). Esto se tendrá que tener en cuenta cuando se lean atributos de más de un byte.

Hoy en día, FAT32 cada vez se utiliza menos, pues ha sido substituido por sistemas de ficheros con mejores prestaciones (*Journaling*[[1]](#footnote-2), almacenamiento de ficheros mayores a 4GB, capacidad para volúmenes mayor a 2 TB, permisos acceso a ficheros, etc.).

De hecho, ya hacía tiempo que no se utilizaba FAT32 en los diferentes sistemas operativos, pero seguía siendo útil en el tema de compatibilidad, pues a diferencia de NFTS, EXT o APFS, era un sistema de ficheros que todos los SO entendían, y poniéndolo (por ejemplo) en un USB, garantizabas que siempre lo pudieras leer. Desgraciadamente, hace un tiempo salió EXFAT, un sistema de ficheros de mejores prestaciones (sobre todo en cuestiones de tamaños de fichero y volumen) y con un rango de compatibilidad similar, de manera que FAT32 ha quedado como *Legacy*.

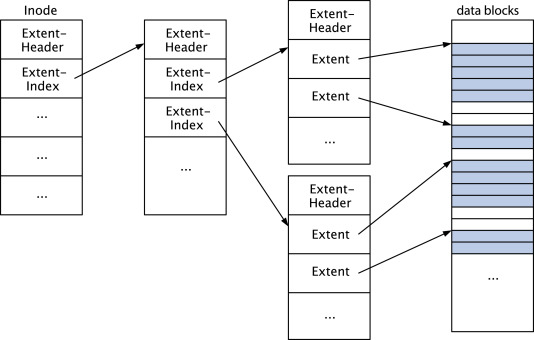
## EXT4

EXT4 se trata de un sistema de ficheros principalmente usado en las distribuciones Linux que incluye *Journaling*.

Este se basa en la distribución del espacio en memoria en bloques de tamaño fijo (suelen ser de 4kB), los cuales son clasificados en bloques de información, la tabla de inodos[[2]](#footnote-3), un bloque con la información para los inodos libres, otro para los bloques de datos libres y uno para la información del sistema de ficheros.

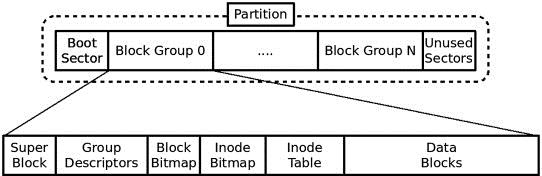
Este sistema de ficheros es una mejora de EXT3 (es compatible con este), puesto que hace un menor uso de la CPU, puede llegar a almacenar ficheros de hasta 16TB y sistemas de ficheros de hasta 1EB[[3]](#footnote-4) y mejora la velocidad de lectura y de escritura (cabe decir que la eliminación de fichero es más lenta).

Cabe destacar que, debido al uso de *Extents* en vez del uso de las *Linked Lists* propias de EXT2 y EXT3, el rendimiento al trabajar con un fichero de grandes dimensiones es mayor, pues solo se hace uso de la primera dirección de memoria seguido de un offset.



La estructura de datos usada para las particiones de EXT4 está formada por dos partes bien diferenciadas:

* Bloque de *Boot*. Este es reservado para cuando se inicia el sistema.
* Bloques de grupo. EXT4 se divide en distintos grupos formados por 2^15 bloques de disco[[4]](#footnote-5), los cuales tienen información sobre el sistema.



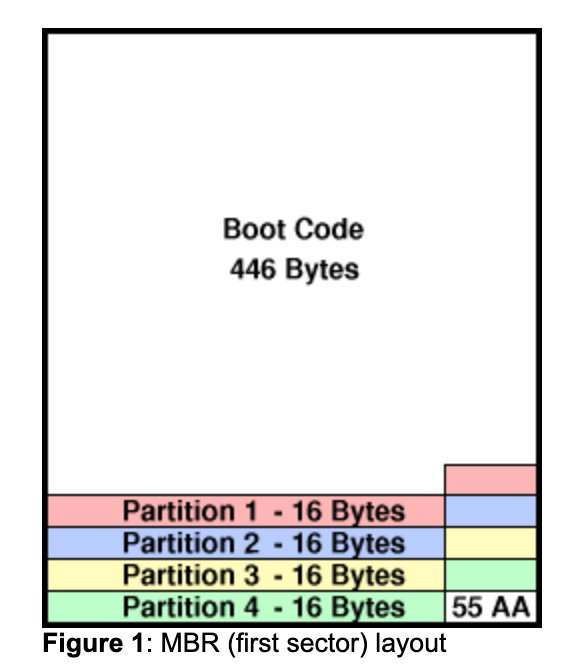
# Investigación

Este apartado contiene datos importantes para la realización de la práctica.

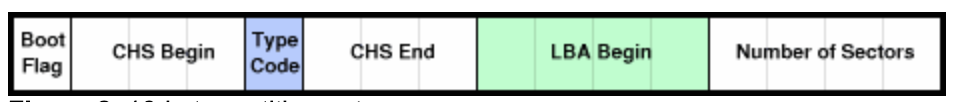
## Detección del sistema de ficheros

### 2.1.1 FATX y EXTX

Si se estuviera trabajando con un sistema con MBR o GPT, con varias particiones, se podría identificar prácticamente todos los tipos de sistemas de ficheros disponibles, pues en la meta-data de cada partición existe un campo TYPE CODE, que corresponde a un tipo de sistema de ficheros.



En la siguiente imagen se puede observar la estructura de una partición:



El problema es que ya se tiene directamente el sistema de ficheros, como si ya se hubiera identificado el tipo y se hubiera encontrado el offset del sector 0 con LBA Begin, de manera que se deberán usar métodos alternativos.

Sabiendo esto, la única manera de mirar los tipos de sistemas de ficheros es mirando los metadatos de estos: sobre los campos con valor fijo por defecto, mirar si este coincide con el del sistema y sobre los otros, mirar si su valor corresponde a algo válido, etc.

Entre los más relevantes, existe el campo *Magic Signature* de EXT4, encontrado en offset 0x38 del Super Block con valor 0xEF53, el cual será igual para todos los sistemas EXTX.

Debido al posicionamiento y longitud del valor (2 bytes), se podría utilizar esto como un identificador para separar las dos categorías de sistemas de ficheros que se deben saber identificar (Si no es EXT será FAT).

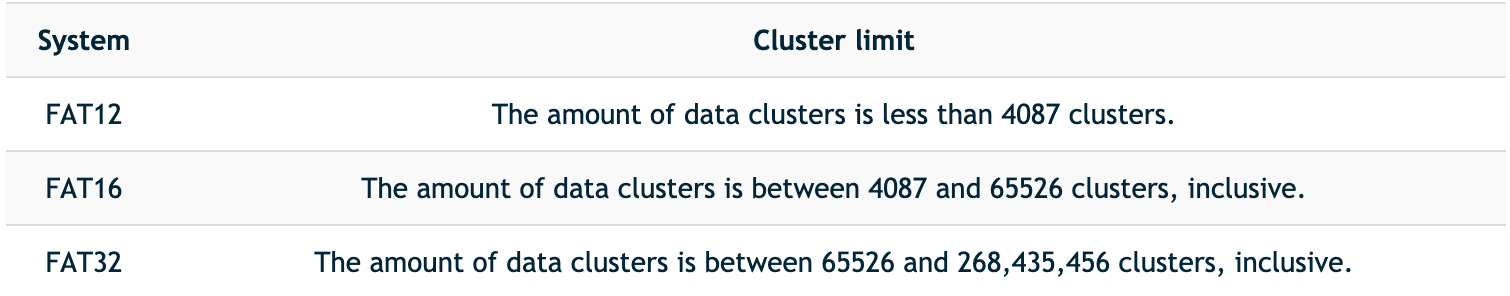
En este tema, la mayoría de los sistemas de ficheros tienen uno o varios campos informativos con traducción válida a ASCII (crean un string con sentido) que indican el tipo de sistema de ficheros. Desgraciadamente, como se ha dicho, estos son campos informativos no utilizados por los sistemas operativos para identificar qué tipo son: se pueden cambiar o crear con valor diferente y no pasaría nada y, por lo tanto, no son fiables.

Volviendo al tema, cabe también saber diferenciar entre diferentes versiones de un mismo tipo de sistema de ficheros, como FAT12, FAT16 y FAT32 (aún más entre FAT12 y FAT16) y EXT2, EXT3 y EXT4.

### 2.1.2 FATX

Diferenciar entre FAT12, FAT16 y FAT32 puede ser básicamente comprobar valores o rangos de estos contra una lista de diferencias hasta que se identifica el sistema de ficheros correcto. Dicho esto, esta solución es complicada, puede dar a dolores de cabeza e incluso llegar a una conclusión incorrecta.

La manera más fiable que se ha encontrado de diferenciar estos tipos de sistemas de ficheros haya lo que haya, es esta:



Para calcular el número de clusters:

El número de clúster es un indicador claro del tipo de FAT, donde las fórmulas para calcularlos son las siguientes:

* Sectores\_de\_data = sectores\_totales – (sectores\_reservados + num\_tablas\_fat \* size\_tabla\_fat)
* Num\_clusters = sectores\_de\_data / sectores\_por\_cluster

### 2.1.3 EXTX

Para poder diferenciar los sistemas de ficheros EXTX entre sí, solo hace falta comprobar que el *Super Block* contenga unos *flags* en concreto en el campo *s\_feature\_compat* (0x064):

* *INCOMPAT\_EXTENTS* (0x040). Se trata de un sistema de ficheros EXT4 si dispone de este *flag*.
* *COMPAT\_HAS\_JOURNAL* (0x004). Se trata de un sistema de ficheros EXT3 si dispone de este *flag* y no del anterior.

Cabe destacar que, si no se dispone de ninguno de los dos *flags* se trata de un sistema de ficheros EXT2.

## Extracción de la meta-data

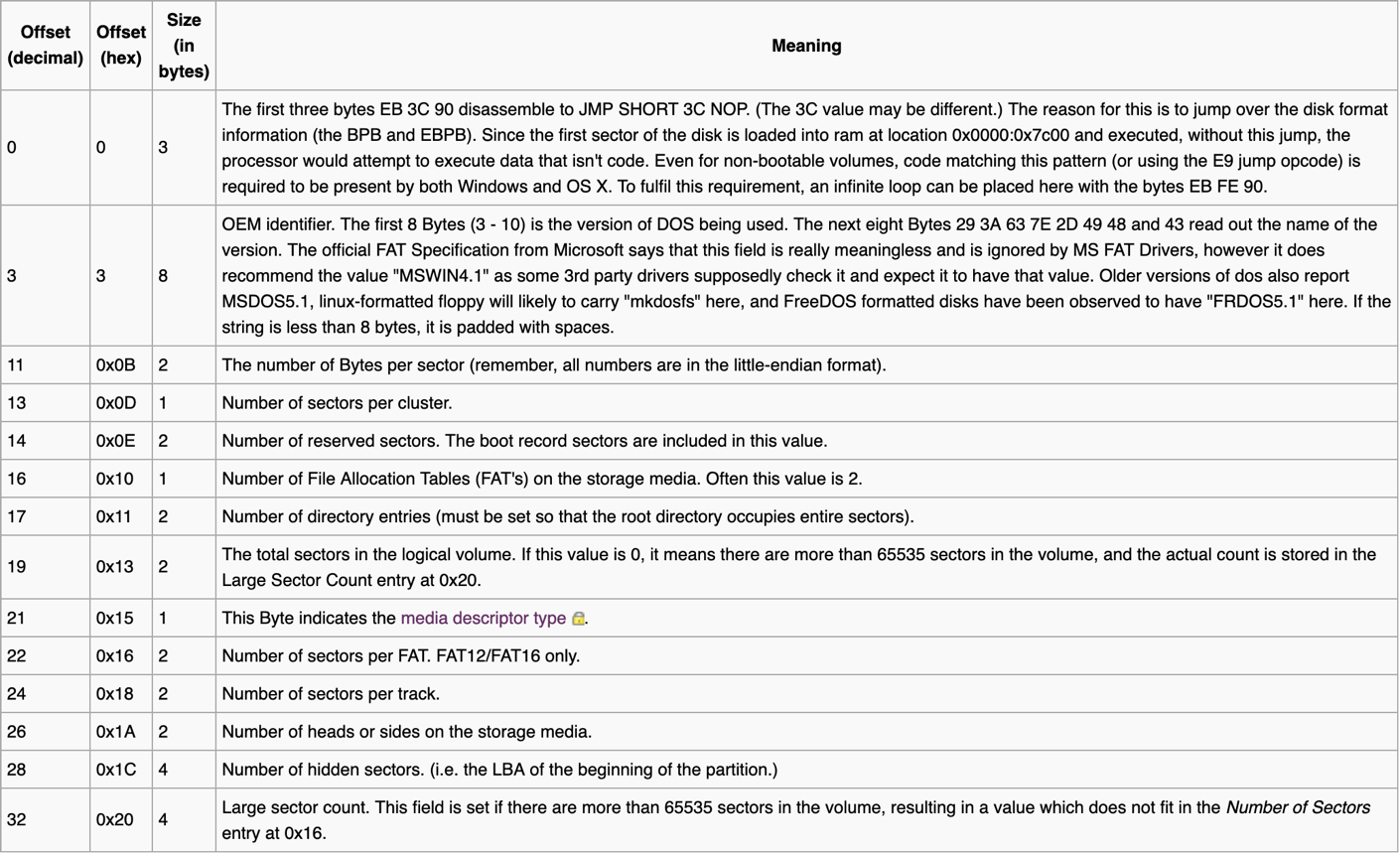
### FAT32

Toda la meta-data del sistema de ficheros FAT32 se encuentra en el primer sector (sector 0 *filesystem*) de este, llamado también *Boot Record*. En FAT, un sector puede ocupar valores de 512, 1024, 2048 o 4096 bytes, aunque en el 99% de los casos suele ser 512B.

#### Boot Record

El boot record se divide en dos partes, una común entre los diferentes sistemas FAT y una particular de cada uno.

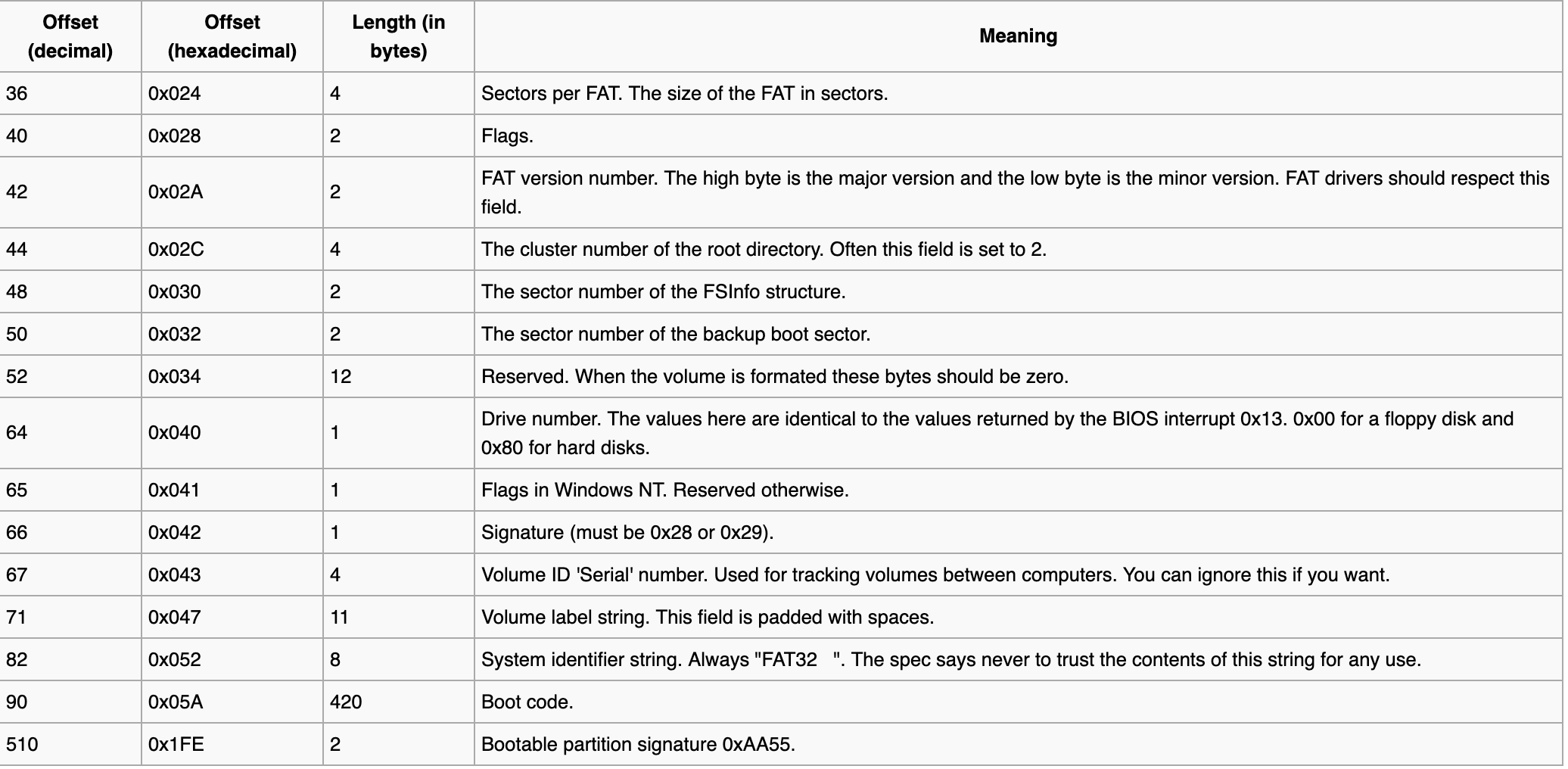
##### BPB (BIOS Parameter Block)

****

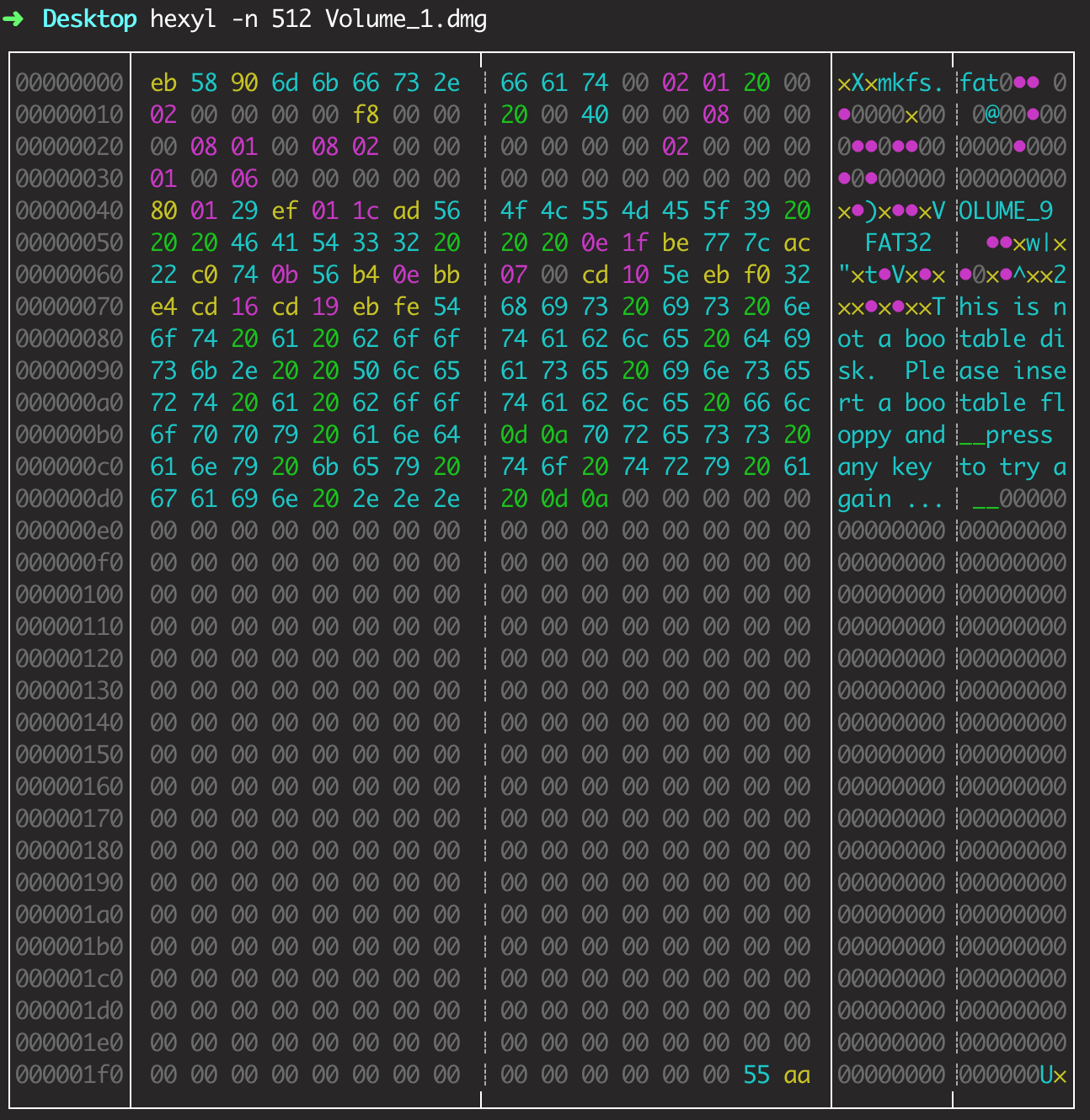
La BPD es común entre FAT12, FAT16 y FAT32.

#### Extended Boot Record (FAT32)

El *Extended Boot Record* diverge entre FAT12, FAT16 y FAT32.



A continuación, se muestra un ejemplo de metadatos de un sistema de ficheros FAT32:



A continuación, se hablará de los metadatos más importantes encontrados en este primer sector. Hay que recordar que cuando se hable de offset, este será respecto al inicio del filesystem o, en nuestro caso, respecto al inicio del fichero.

##### Nombre del volumen - Volume Label String

Offset: 0x47 | Size: 11 bytes

El identificador con el que aparecerá el volumen cuando se monte.

Ejemplo: 56 4f 4c 55 4d 45 5f 39 – VOLUME\_9



##### Tamaño del sector – Bytes per sector

Offset: 0x0B | Size: 2 bytes

El número de bytes por sector. Como se ha dicho antes, estos pueden ser 512, 1024, 2048 o 4096 bytes, si bien normalmente es 512B.

Es importante para el cálculo del tamaño de clúster.

Ejemplo: 0x00 0x02 (*Little Endian*) 0x02 0x00 (*Big Endian*) – 512 bytes por sector.

##### Sectores por clúster - Sectors Per Cluster

Offset: 0x0D | Size: 1 byte.

El número de sectores por clúster (cada clúster es representado por un entero de 32 bits en la FAT). Este campo puede tomar los siguientes valores: 1, 2, 4, 8, 16, 32, 64 o 128.

Permite, junto con el tamaño de sector, saber el espacio que ocupa un clúster, la unidad mínima de almacenamiento de FAT32.

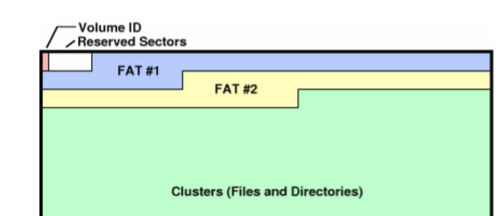
Ejemplo: 0x01 – 1 sector por clúster.

##### Sectores reservados – Number of Reserved Sectors

Offset: 0x0E | Size: 2 bytes.

Número de sectores reservados, los cuales se encuentran entre el sector 0 (*Boot Record*) y la tabla FAT. Normalmente son 32 sectores.

Permite, junto con el número de tablas FAT y su tamaño, saber dónde empiezan los clústeres.



Ejemplo: 0x20 – 32 sectores reservados.

##### Número de FATs – Number of FATs

Offset: 0x10 | Size: 1 byte.

Número de tablas FAT (se usa la primera y las otras son de *backup* por si la primera se corrompe). Normalmente este valor es 2.

Junto con los sectores reservados y el tamaño de tabla, es importante para saber dónde empiezan los clústeres.

Ejemplo: 0x02 – 2 tablas FAT.

##### Primer clúster de root - Root Directory First Cluster

Offset: 0x2C | Size: 4 bytes. (32 bits).

Indica el primer clúster del directorio root. Normalmente es 2. Tener en cuenta que el valor dado es desde el primer clúster (se recomienda observar la captura del apartado de sectores reservados), no desde el principio del sistema de ficheros.

Todo el acceso al sistema de ficheros será desde este punto inicial

Ejemplo: 02 00 00 00 (*Little Endian*) 00 00 00 02 (*Big Endian*) – clúster número 2.

##### Sectores por FAT – Sectors per FAT

Offset: 0x24 | Size: 4 bytes. (32 bits).

Número de sectores que ocupa la tabla FAT. Junto con el número de tablas y los sectores reservados, permite saber dónde empiezan los clústeres.

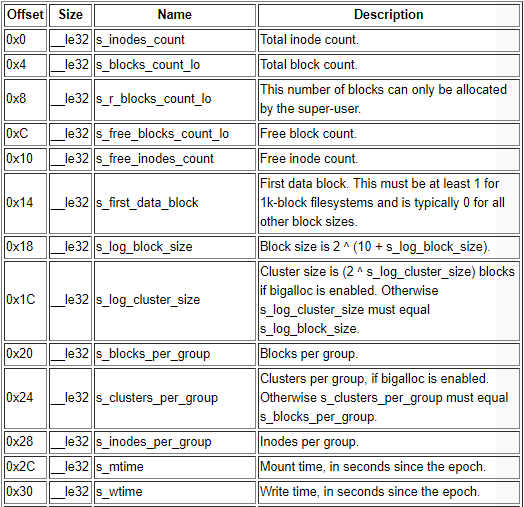
Ejemplo:08 02(*Little Endian*) 02 08 (*Big Endian*) – 520 sectores.

### EXT4

La meta-data del sistema de ficheros se encuentra en el *Super Block*, el cual está situado en un offset de 1024 bytes (espacio reservado para el *Boot Sector*), siendo localizado en principalmente en el *block group* 0 (a no ser que el tamaño de bloque sea de 1024 bytes, localizándose en el *block group* 1).

Cabe destacar que el Super Block puede encontrarse repetido en el sistema de ficheros siempre y cuando estuviera establecido el *flag* *RO\_COMPAT\_SPARSE\_SUPER* (0x0001) en el campo *s\_feature\_ro\_compat* (0x064), de manera que se encontraría en los grupos que son potencia de 3, 5 y 7 (a parte del grupo 0).

A continuación, se mostrará algunos de los campos asociados al *Super Block*.



#### Tamaño de inodo

Corresponde al campo *s\_inode\_size* (0x058), el cual tiene un tamaño de 16 bits.

#### Número de inodos

Corresponde al campo *s\_inodes\_count* (0x000), el cual tiene un tamaño de 32 bits.

#### Primer inodo

Corresponde al campo *s\_first\_ino* (0x054), el cual tiene un tamaño de 32 bits.

#### Grupo de inodos

Este campo no se ha sabido identificar con ninguna estructura de EXT4.

#### Inodos libres

Corresponde al campo *s\_free\_inodes\_count* (0x010), el cual tiene un tamaño de 32 bits.

#### Tamaño de bloque

Corresponde a un cálculo que requiere del campo *s\_log\_block\_size* (0x018), el cual tiene un tamaño de 32 bits.

La fórmula es la siguiente:

#### Bloques reservados

Corresponde al conjunto de dos campos, puesto que tiene un total de 64 bits, siempre y cuando el campo *s\_feature\_incompat* (0x064) no tenga el valor *INCOMPAT\_64BIT* (0x0080), sino se usará el de menor peso.

Estos campos son: *s\_r\_blocks\_count\_lo* (0x008, menor peso) y *s\_r\_blocks\_count\_hi* (0x154, mayor peso).

#### Bloques libres

Corresponde al conjunto de dos campos, puesto que tiene un total de 64 bits, siempre y cuando el campo *s\_feature\_incompat* (0x064) no tenga el valor *INCOMPAT\_64BIT* (0x0080).

Estos campos son: *s\_free\_blocks\_count\_lo* (0x00C, menor peso) y *s\_free\_blocks\_count\_hi* (0x158, mayor peso).

#### Primer bloque

Corresponde al campo *s\_first\_data\_block* (0x014), el cual tiene un tamaño de 32 bits.

#### Block group

Corresponde al campo *s\_block\_group\_nr* (0x05A), el cual tiene un tamaño de 16 bits.

#### Frags group

Este campo no se ha sabido identificar con ninguna estructura de EXT4.

#### Nombre de volumen

Corresponde al campo *s\_volume\_name* (0x078), el cual tiene un tamaño de 16 carácteres.

#### Última comprobación

Corresponde al campo *s\_lastcheck* (0x040), el cual tiene un tamaño de 32 bits. Cabe destacar que el tiempo es relativo a la era Unix (1970-01-01 00:00:00 GMT).

#### Último montaje

Corresponde al campo *s\_mtime* (0x02C), el cual tiene un tamaño de 32 bits. Cabe destacar que el tiempo es relativo a la era Unix (1970-01-01 00:00:00 GMT).

#### **Última escritura**

Corresponde al campo *s\_wtime* (0x030), el cual tiene un tamaño de 32 bits. Cabe destacar que el tiempo es relativo a la era Unix (1970-01-01 00:00:00 GMT).

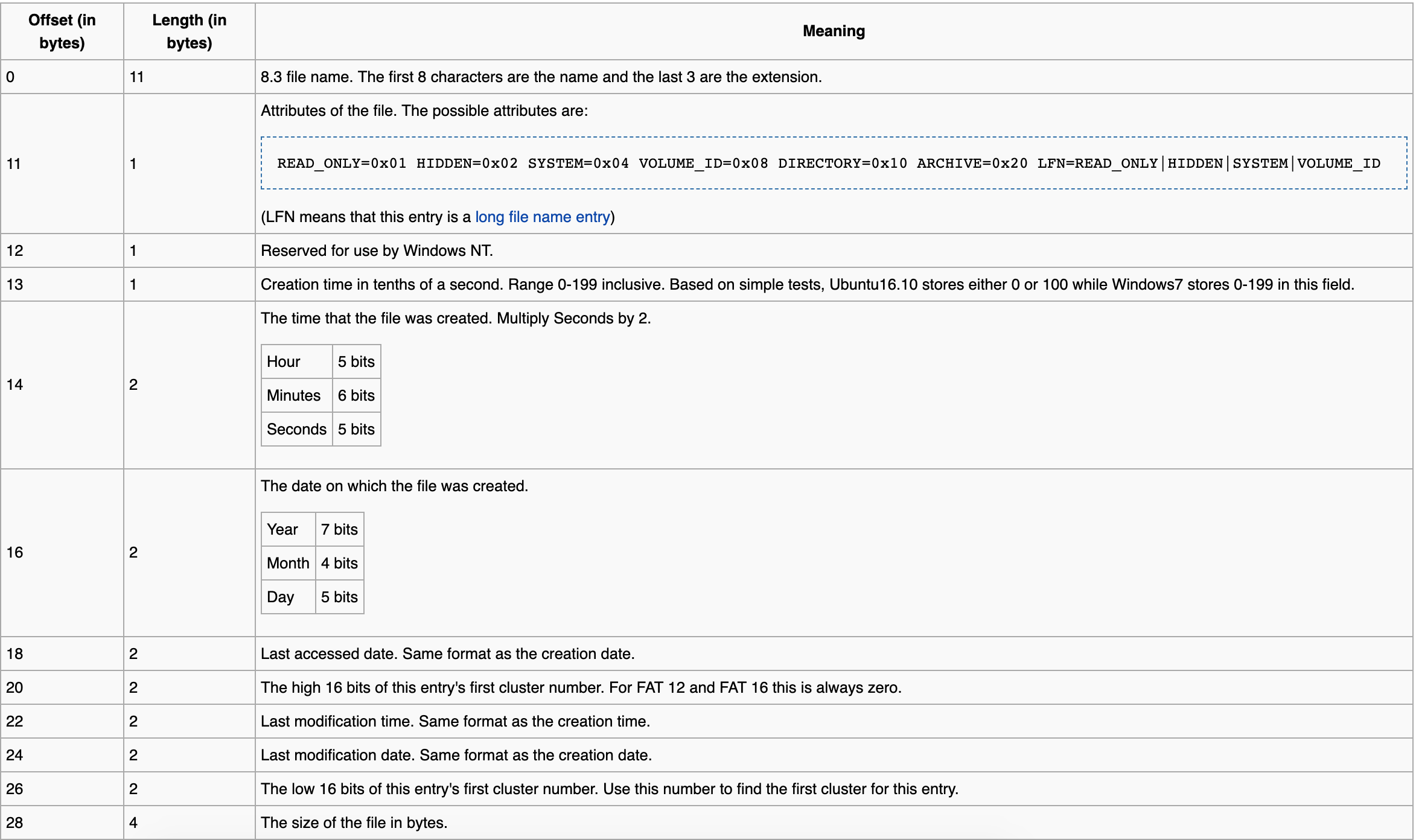
## Localización de un fichero y meta-data

### FAT32

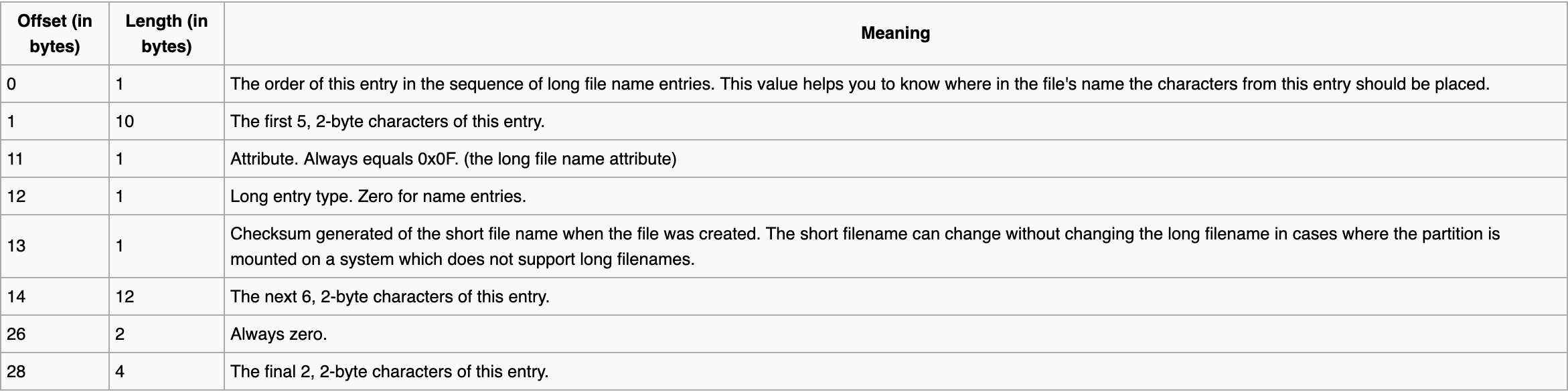
En FAT32, la meta-data de un fichero se encuentra en el directorio (fichero especial). Cada directorio se estructura en entradas de 32 bits, las cuáles pueden ser ficheros (directorios también son ficheros), entradas no usadas o fin de directorio.

En el caso del fichero, dependiendo de la longitud del nombre de este, se puede estructurar a lo largo de 1 o 2 entradas de 32 bits.

Todos los ficheros tienen una entrada en formato 8.3 (8 bytes name – 3 bytes *extension*).



Si el nombre de fichero es mayor a 8 bytes, la entrada del fichero tiene previo a entrada formato 8.3 esta entrada:



Si el fichero tiene un nombre largo, de los atributos se utiliza la parte de LFN.

Respecto a los metadatos, se puede observar, notablemente, cuándo se creó el fichero, cuándo se modificó, si solo se puede leer o está escondido, su tamaño, etc.

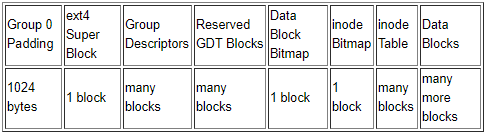
En los otros casos, si la entrada no se usa, el primer byte de esta se leerá a 0xE5, y si es fin de directorio, el primer byte de la entrada será 0.

Consecuentemente, recorrer el sistema de ficheros supondrá, desde el primer directorio ir accediendo, como si fuera un árbol, a las diferentes entradas de ficheros que sean directorios de forma recursiva.

Finalmente, hace falta tener en cuenta que si se llega al final de un clúster y aún no se ha encontrado la entrada que indica que se ha terminado el directorio (0), es porque un directorio puede ocupar más de un clúster.

### Ext4

Con tal de poder localizar un fichero, se tendrá que acceder al Block Group Descriptor para poder acceder a la tabla de inodos, puesto que este bloque contiene la información de dónde se localiza esta.



Los campos que se consultarán son: *bg\_inode\_table\_lo* (0x08, menor peso) y *bg\_inode\_table\_hi* (0x28, mayor peso).

Cabe destacar que se usaran ambos campos siempre y cuando el campo *s\_feature\_incompat* (0x064) del Super Block no tenga el valor *INCOMPAT\_64BIT* (0x0080), sino se usará el de menor peso.

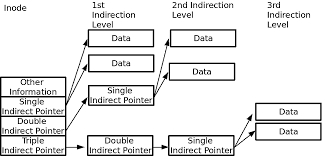
Una vez accedida a la estructura de datos de la tabla de inodos, lo primero a revisar es si el inodo que se está consultando es un directorio o un fichero para saber si la información de la tabla es del fichero o del directorio.

Si no se tratara de un fichero, se consultaría el campo de *i\_block* (0x28), el cual dispondría de la estructura de datos que mapearía los datos del inodo, el cual tendría un número de bloques que se calcularía según si el flag *EXT4\_HUGE\_FILE\_FL* (0x00040000) estuviera presente en el campo *i\_flags* (0x20) y de si el Super Block usa el flag *RO\_COMPAT\_HUGE\_FILE* (0x0008) en el campo *s\_feature\_ro\_compat* (0x64):

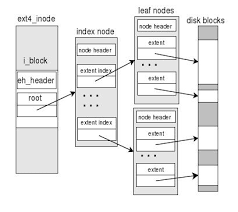
* Sin ningún flag:
* Con *RO\_COMPAT\_HUGE\_FILE* y sin *EXT4\_HUGE\_FILE\_FL*:
* Con *RO\_COMPAT\_HUGE\_FILE* y con *EXT4\_HUGE\_FILE\_FL*:

#### Localización del inodo fichero

Dependiendo de si el flag *EXT4\_EXTENTS\_FL* (0x00080000) está presente en *i\_flags* (0x20), el inodo usará un direccionamiento directo/indirecto para tener compatibilidad con Ext2 y/o Ext3 si no está presente.



En el caso que el flag estuviera presente, se usará un Extent Tree (Hash Tree como aproximación).



Una vez llegados a las hojas o casilla de la Linked List, analizaríamos la entrada del directorio para obtener el inodo que contiene su información gracias al campo *inode* (0x0) en una directorio lineal si el flag *EXT4\_INDEX\_FL* (0x00010000) está presente en el campo *i\_flag* (0x20) o gracias al campo *dot.inode* (0x00) en un árbol de directorios.

Antes de seguir buscando en el inodo de manera recursiva o de comprobar el tipo del inodo, deberíamos de acceder al campo *name* (0x08) y *name\_len* (0x06) para comprobar si tiene el mismo nombre que el que se desea, terminando con la búsqueda en esa rama (se pueden buscar más ficheros con el mismo nombre dependiendo de la implementación).

#### Obtención de la meta-data

Para obtener la meta-data del fichero, con consultar los datos de los campos *i\_crtime* (0x90, fecha de creación) y *i\_crtime\_extra* (0x94, precisión mayor y adicional al anterior campo) para obtener la fecha de creación del fichero.

Por lo que respecta al tamaño, bastaría con la consulta de los campos *i\_size\_lo* (0x04, menor peso) y *i\_size\_high* (0x6C, mayor peso).

## Obtención de los datos de un fichero

### FAT32

Suponiendo que se tiene el directorio del fichero y por lo tanto la meta-data de este (entrada 32 o 64 bits), obtener los datos será muy sencillo. Como ya se ha explicado en el apartado 1.1, en la entrada del fichero se encuentra la dirección del primer clúster del fichero. Este corresponde a la vez unos X bytes en el espacio de clústeres, que son parte de los datos del fichero, y a la vez corresponde a una entrada de la tabla FAT, que indicará si hay otro clúster para ese fichero o ya se ha leído toda la información (mayor o igual a 0x0FFFFFF8).

Aunque tal vez se debería haber explicado en un apartado anterior, la manera en que se encuentra la asociación entre en número de clúster y la entrada de la FAT es la siguiente:

* Fat\_offset (en bytes) = num\_cluster \* 4 (entradas 4 bytes)
* sector\_fat = sector\_0 (boot) + sectores\_reservados + (fat\_offset / bytes\_sector)
* entrada\_sector\_fat = fat\_offset % bytes\_sector

Con esto se consigue el sector de la FAT y el valor correspondiente dentro del sector. Si se quisiera acceder a la segunda tabla de FAT, solo haría falta añadir el número de sectores que ocupa una tabla en el cálculo del sector de la FAT.

### Ext4

Partiendo de la idea de la localización del fichero, bastaría con indagar en cada uno de los inodos e ir obteniendo la información que está siendo apuntada por los inodos y mostrarla por pantalla.

Cabe destacar que tendremos que tener en cuenta el tamaño del fichero con tal de no mostrar bytes de más del último inodo que está siendo apuntado.

# Estructuración de la práctica

## 3.1 Módulos y funciones que se usarán

**Módulo de validación de parámetros**

Este módulo se trata de un preprocesador que se encargará de comprobar que la información ofrecida por el usuario sea válida a nivel de: número de parámetros, y la existencia de las operaciones.

**Módulo pantalla**

Este módulo se encargará de mostrar la información de interés para el usuario, de manera que nadie más tendrá una interacción directa con el usuario.

**Módulo identificador**

Este módulo se encargará de identificar qué tipo de sistema de ficheros se trata (FAT32 o Ext4). En el caso de que no se identificará ninguno, se mostraría un error.

Cabe destacar que identificará los sistemas FATX y EXTX, de manera que tendrá una función para comprobar si se trata de un sistema de ficheros EXTX y sino comprobará si es FATX.

**Módulo meta-data**

Este módulo estará subdividido en dos módulos o conjunto de operaciones según se trate de un sistema de ficheros FAT32 o EXT4.

Este módulo se encargará de obtener la información del sistema de ficheros y la devolverá (un array de strings, por ejemplo).

Debido a que los sistemas de ficheros utilizan métodos completamente diferentes para estructurar la información y por consiguiente también para acceder a ella, se ha creado módulos especializados para cada uno de ellos:

**Módulo inodo (EXT4)**

Este módulo se encargará de varias funcionalidades relacionadas con la búsqueda, obtención de meta-data y de la información de un fichero en el sistema de ficheros EXT4

Sus funciones se encargarían de:

* Devolver el inodo en el caso de buscar el fichero.
* Obtener la meta-data del inodo especificado.
* Obtener la información que es referenciado por el inodo especificado.

**Módulo FAT (FAT32)**

Este módulo se encargará de varias funcionalidades relacionadas con la búsqueda, obtención de meta-data y de la información de un fichero en el sistema de ficheros FAT32.

Tendrá funcionalidades para:

* Buscar un fichero (también directorio) en el sistema de ficheros.
* Devolver la meta-data del fichero.

1. Se basa en llevar un registro diario de cierta información para poder recuperar los datos de un sistema de ficheros que haya sufrido algún error. [↑](#footnote-ref-2)
2. Contiene la meta-data de los ficheros. [↑](#footnote-ref-3)
3. 1 EB = 1024 PB = 2^50 B [↑](#footnote-ref-4)
4. Unidad en que se divide el disco para almacenar información, ya sea del sistema de ficheros o de los ficheros a almacenar. [↑](#footnote-ref-5)