CENTRO UNIVERSITARIO
DE TECNOLOGÍA
Y ARTE DIGITAL

Sistemas Operativos

5. Gestión de ficheros

Sergio Bermúdez Fernández
sergio.bermudez@u-tad.com
Miguel Angel Mesas
miguel.mesas@u-tad.com
Carlos M. Vallez
carlos.vallez@u-tad.com

2024-2025

Introducción

Todos los equipos necesitan un sistema de almacenamiento no volátil para guardar datos y programas. Primero fueron las cintas magnéticas, las unidades de lectura con los tambores girando crearon la imagen clásica del Centro de Proceso de Datos. En los 80, en forma de casette, entraron en los hogares con equipos como el ZX Spectrum.

Introducción

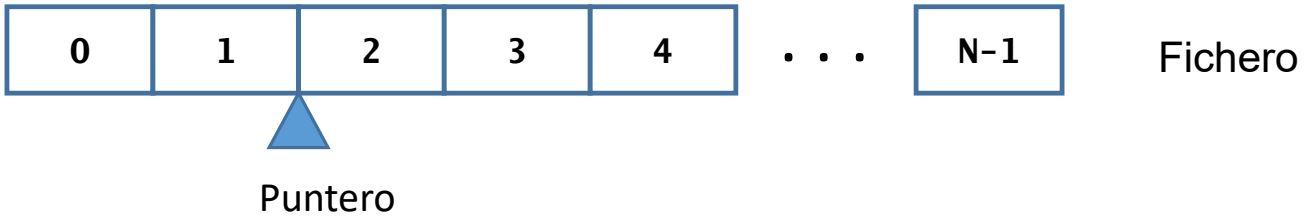
El rey del almacenamiento secundario no volátil ha sido el disco magnético, durante más de medio de siglo (el primer modelo comercial IBM 350 se lanzó en 1956(. Su principal ventaja sobre la cinta era que la lectura/escritura ya no tenía que ser secuencial y su velocidad era mucho mayor.

Introducción

Hoy los dispositivos de estado sólido han desplazado a los discos magnéticos en los ordenadores personales y se usan otro tipo de dispositivos como tarjetas de memoria SD o USB cuya física y funcionamiento no tienen nada que ver con los de su antepasado. Sin embargo, los programas de usuario no necesitan conocer estos detalles y son inmunes a los cambios de tecnología porque usan una abstracción muy exitosa, el **fichero**. No hay que confundir el fichero como abstracción con su soporte físico, el gran acierto de Unix fue desacoplar estas dos acepciones. En Unix, prácticamente todo se puede tratar como un fichero.

Fichero

Un fichero es una abstracción del almacenamiento secundario, una sucesión de bytes cuya semántica no afecta al Sistema Operativo. La información se presenta como una serie de registros consecutivos del mismo tamaño, con un puntero que permite acceder de forma aleatoria a cualquiera de ellos.



Esta abstracción oculta los detalles del soporte físico. Un fichero tiene que cumplir 3 condiciones:

1. Almacenar una cantidad de información ilimitada (en comparación con la RAM)
2. Sobrevivir al proceso que lo creó
3. Permitir el acceso simultáneo de varios procesos

Fichero

```c
#include <stdio.h>

int main()
{
    FILE *fentrada;
    int bytesread;
    typedef struct {
        char nombre[12];
        int nBTC;
    } T_BTC;

    T_BTC regBTC;

    fentrada = fopen("Bitcoins.dat","rb");
    if (fentrada != NULL)
    {
        while (( bytesread = fread(&regBTC,sizeof(T_BTC),1,fentrada)) > 0)
          printf("Nombre: %s, bitcoins: %d\n", regBTC.nombre, regBTC.nBTC);
    }
    fclose(fentrada);

}
```

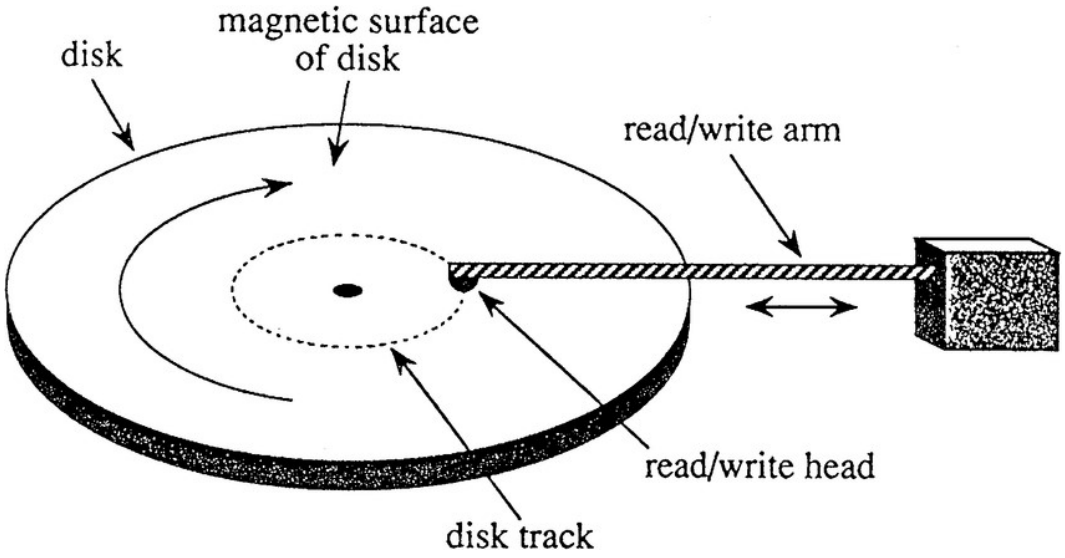
En este ejemplo, el programa lee de forma secuencial los registros de un fichero binario y muestra su contenido por pantalla. La función `fread()` devuelve el número de bytes leídos cada vez que se invoca

El puntero se puede mover con las funciones `fseek()` que lo coloca al inicio del registro solicitado o `rewind()`, que lo devuelve al inicio. Son un recuerdo del acceso secuencial de las cintas magnéticas.

Fichero

Durante medio siglo el disco magnético ha sido el medio de almacenamiento secundario por excelencia. Sus detalles de funcionamiento se reflejan en las funciones del Sistema Operativo relacionadas con los ficheros, por lo que resulta imprescindible conocerlos.

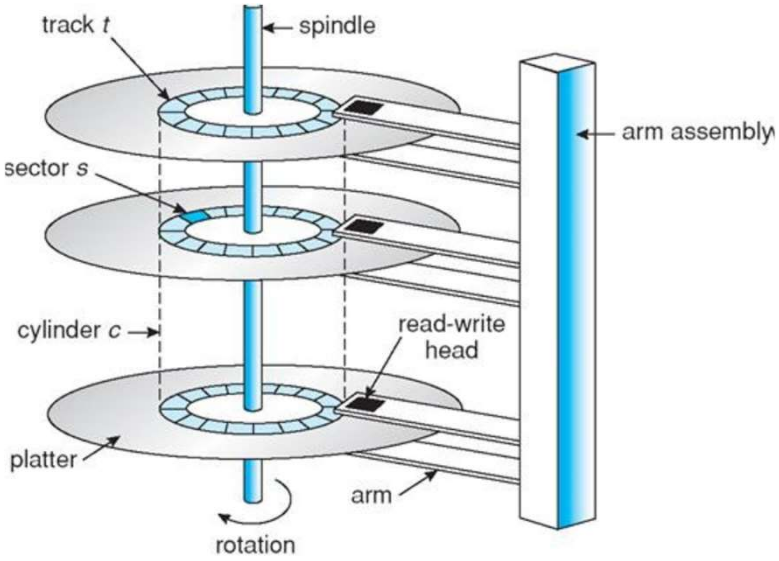
Un disco magnético es una superficie giratoria sobre la que se ha depositado un material ferromagnético, que puede ser escrito y leído con un electroimán. Según el estado de imantación se distinguen el 0 del 1.



Fichero

La unidad mínima de lectura/escritura del disco magnético es el **sector**, típicamente 512 bytes. Esto quiere decir que la cabeza no puede leer o modificar un byte particular sino todo el conjunto.

Un conjunto de sectores concéntricos se denomina **pista**. A media que fue mejorando la tecnología se añadieron varios discos en el mismo eje con sus cabezas lectoescritoras correspondientes. Todas las pistas de igual diámetro de las distintas superficies constituyen un cilindro.



Para leer o escribir un sector, la cabeza se desplaza radialmente hasta situarse sobre la pista y allí espera que pase el sector deseado, lo que ocurre en poco tiempo porque el eje gira a una gran velocidad. Un preámbulo en cada sector permite identificar a la cabeza el número del sector que está transitando en ese instante.

Fichero

La unidad mínima de transferencia entre el disco y el ordenador es el **bloque**. Es la cantidad de información más pequeña que viajará entre el driver y el controlador del disco. En los sistemas con memoria virtual se hace coincidir con el tamaño de página, así que en Windows y Linux es de 4kB, es decir 8 sectores. La traducción de número de bloque a dirección física (cilindro, pista, sector) se realiza en el driver [tema 6]. En algunos sistemas se denomina **agrupación** (*cluster* en inglés) a un conjunto de bloques, en Unix agrupación es sinónimo de bloque.

La estructura de un fichero consiste en el mapa de agrupaciones que ocupa, es lo que se denomina **mapa del fichero** y lo gestiona el Sistema Operativo. Las agrupaciones de un fichero, por regla general, no serán contiguas.

El hecho de que un fichero ocupe agrupaciones completas produce un problema de **fragmentación interna**, hay espacio desaprovechado. Este inconveniente es muy pequeño con los dispositivos actuales, dada su capacidad.

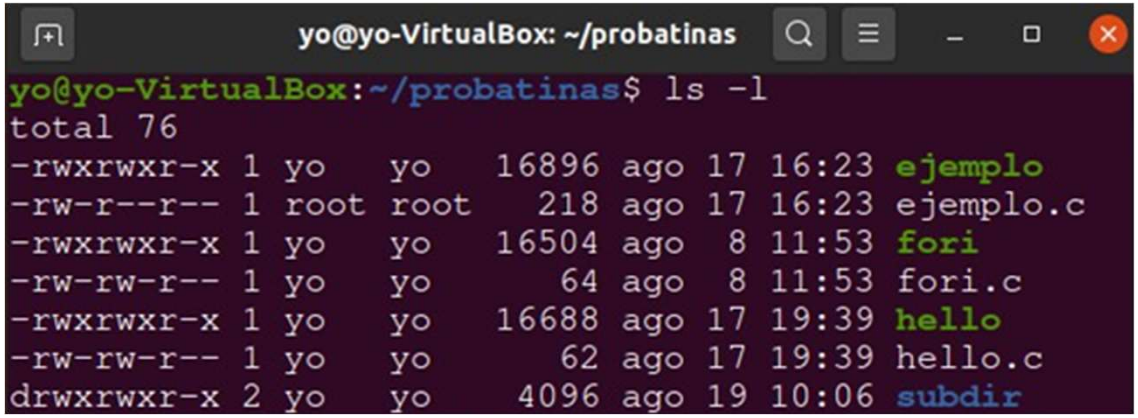
Un problema de la organización en bloques es que cualquier modificación requiere la transferencia de 4096 bytes, aunque solo se modifique uno de ellos. Este efecto se ve compensado porque los drivers están optimizados para esa cantidad, que es la información de una página de memoria virtual.

Fichero

El fichero no consiste únicamente en la información que almacena. Hay un conjunto de metadatos que resulta imprescindible para su manejo:

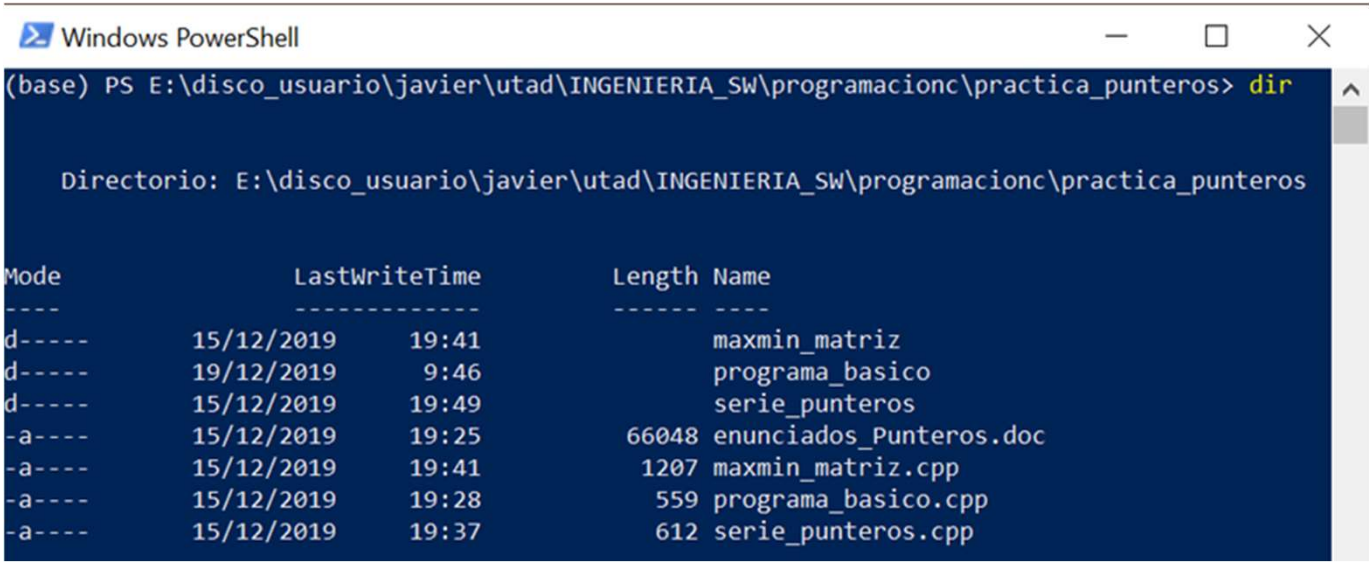
- **Identificador**. Clave única que identifica al fichero. No es la información con la que suele trabajar el usuario, que utiliza el nombre. El nombre es un alias del identificador y usando links se pueden usar varios nombres para un mismo fichero.
- **Nombre del fichero**. Cadena de caracteres que debe ser única dentro de un mismo directorio. En Windows la extensión indica su naturaleza (por ejemplo .EXE para ejecutables). En Unix la extensión es informativa.
- **Fechas** de creación, último acceso y modificación.
- **Tipo de fichero**. regular, de entrada/salida, directorio, etc.
- **Mapa del fichero**. Ya mencionado, es la lista de agrupaciones ocupadas por el fichero.
- **Propietario**. En Unix se identifica por el User ID (UID) y Group ID (GID).
- **Protección**. Bits de permisos del fichero.
- **Tamaño**. Bytes de información útil del fichero, aunque en el soporte físico ocupe un múltiplo entero del tamaño de bloque.

Fichero



El comando `ls` ofrece los metadatos del directorio actual en Linux

En Windows se heredó el comando `dir` de MS-DOS. En *Powershell* se puede invocar con ese nombre o también como `ls`

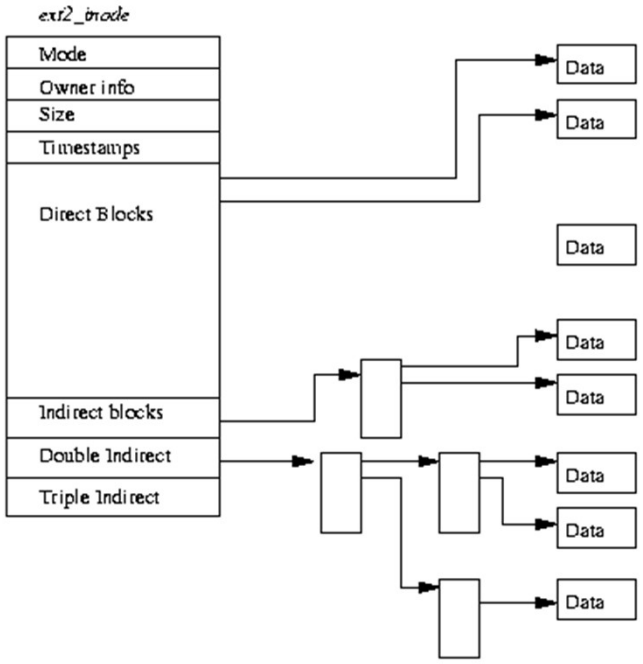


Fichero

La información de metadatos no se guarda en el propio fichero sino en unas estructuras especiales: directorios y Descriptores Físicos de Fichero (DFF). En Windows, el DFF se llama Master File Table (MFT), en Unix i-node (index node) o i-nodo en castellano. La información reside en el soporte físico y se instancia en memoria cuando un programa accede.

i-nodo en un sistema de ficheros Unix ext2. El i-nodo se identifica por un número único que es también el identificador del fichero, guarda datos sobre seguridad y timestamp y es la raíz de un árbol que contiene los números de agrupación que ocupa el fichero.

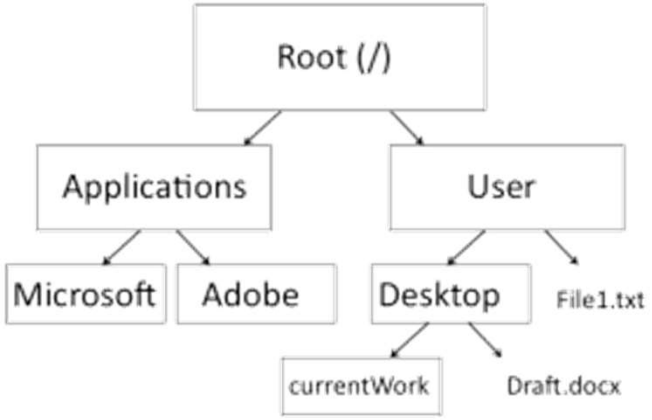
Es importante quedarse con la idea de que la relación i-nodo / fichero es 1 a 1.



Directorio

El directorio es una clase de fichero especial, que guarda información sobre otros ficheros. En un directorio encontramos la relación entre el nombre de un fichero y su DFF. En los Sistemas Operativos antiguos como CP/M o las primeras versiones de MS-DOS el directorio era plano. Su gestión era sencilla pero no eran escalables. cualquier ordenador personal hoy tiene cientos de miles de ficheros, resultaría insoportable para el usuario tener que trabajar con una carpeta única de ese tamaño.

Los subdirectorios permiten agrupar ficheros de características similares. En lugar de una lista, los directorios actuales son estructuras jerárquicas (árboles) que apuntan a ficheros convencionales o a subdirectorios.



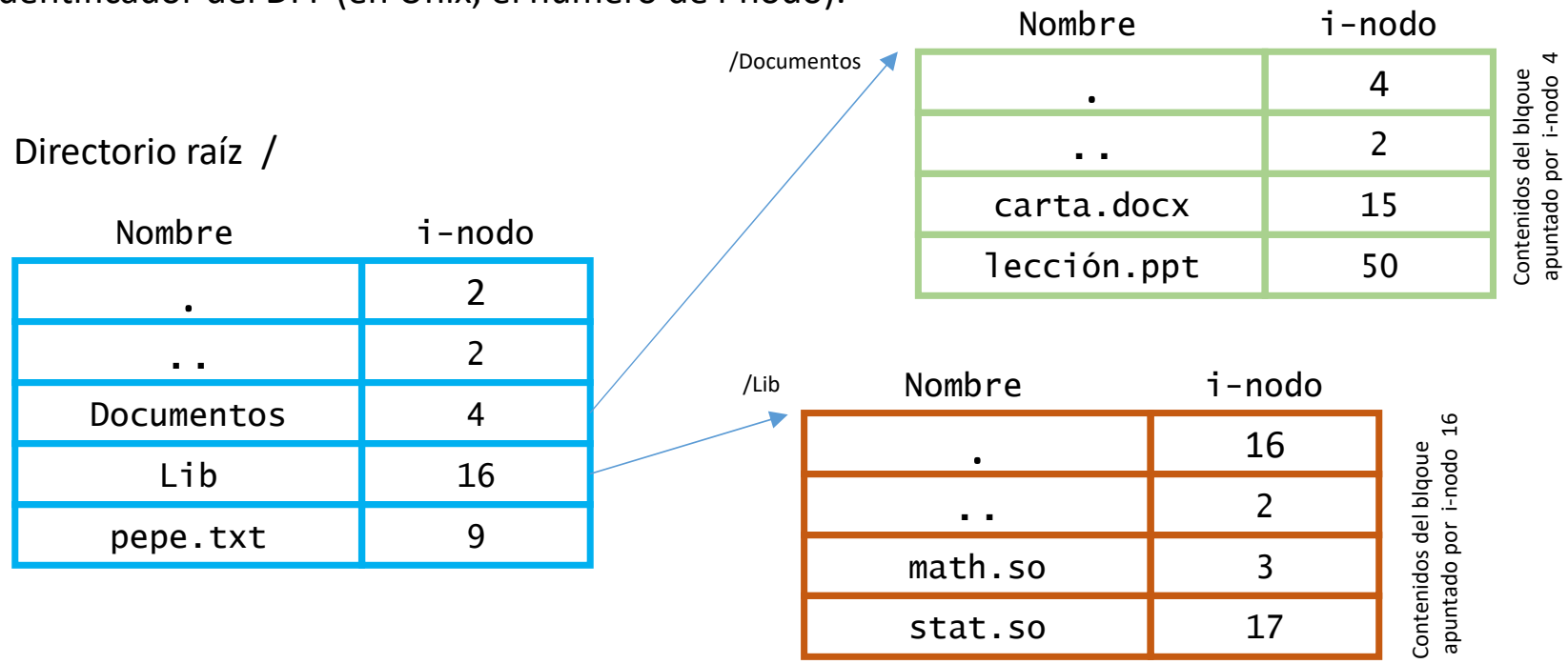

```
struct ext2_dir_entry_2 {
   u32 inode;                  // inode number; count from 1, NOT 0
   u16 rec_len;                // this entry's length in bytes
   u8  name_len;               // name length in bytes
   u8  file_type;              // not used
   char name[EXT2_NAME_LEN]; // name: 1-255 chars, no ending NULL
};
```

Entrada de directorio en ext2

Directorio

Cada fichero recibe un nombre local que tiene que ser único dentro de su carpeta, el nombre absoluto se obtiene anteponiendo el path completo, por ejemplo `/usr/bin/gcc`

La función del directorio es mantener el árbol de descriptores y por tanto se construye con esa estructura. Cada entrada del directorio tiene sólo dos datos, el nombre del fichero y el identificador del DFF (en Unix, el número de i-nodo).



Directorio

Directorio raíz /

Nombre	i-nodo
.	2
..	2
Documentos	4
Lib	16
pepe.txt	9

/Documentos

Nombre	i-nodo
.	4
..	2
carta.docx	15
lección.ppt	50

Contenidos del blqoue apuntado por i-nodo 4

/Lib

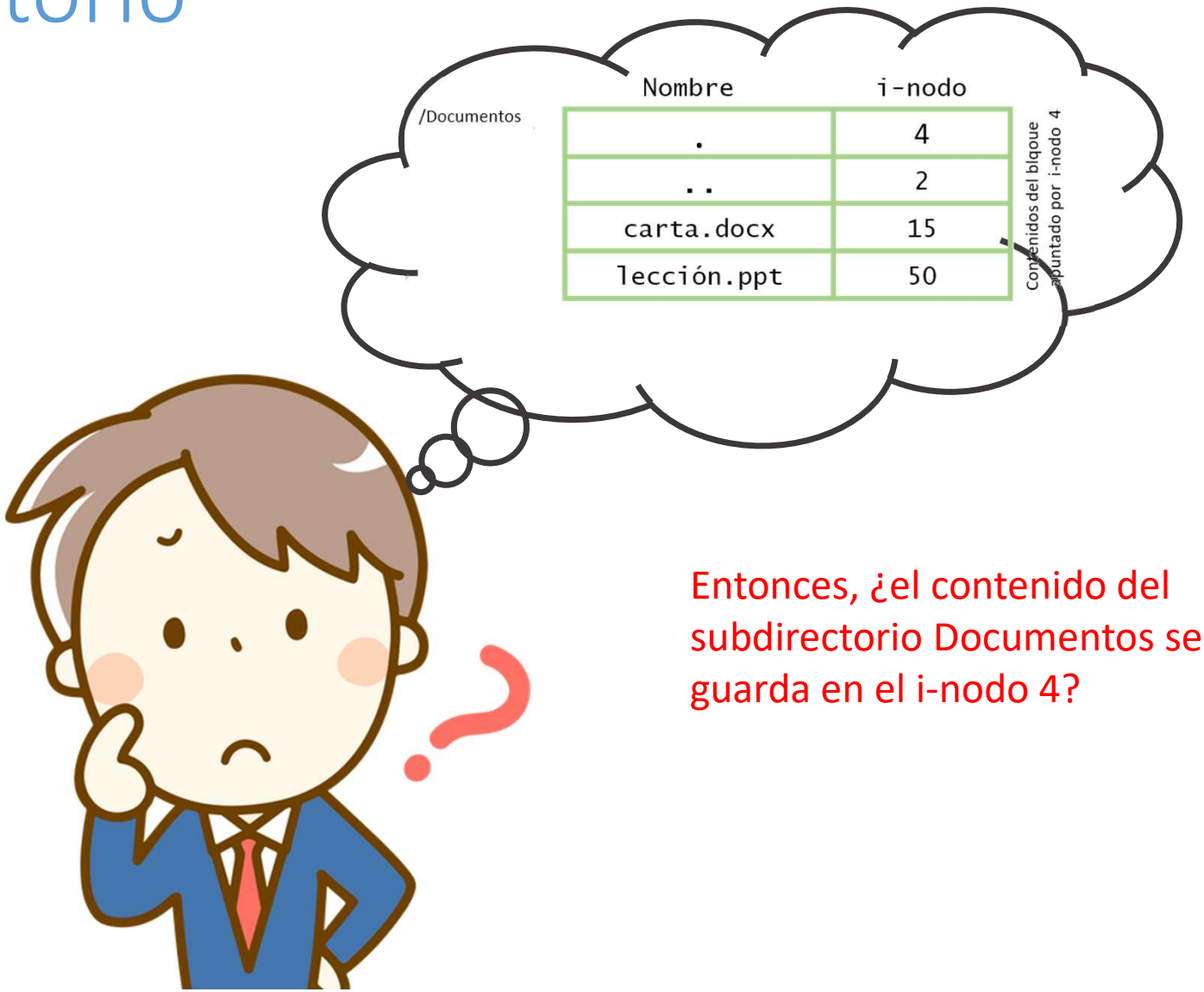
Nombre	i-nodo
.	16
..	2
math.so	3
stat.so	17

Contenidos del blqoue apuntado por i-nodo 16

Cada directorio tiene dos entradas especiales, que apuntan al directorio padre y a sí mismo.

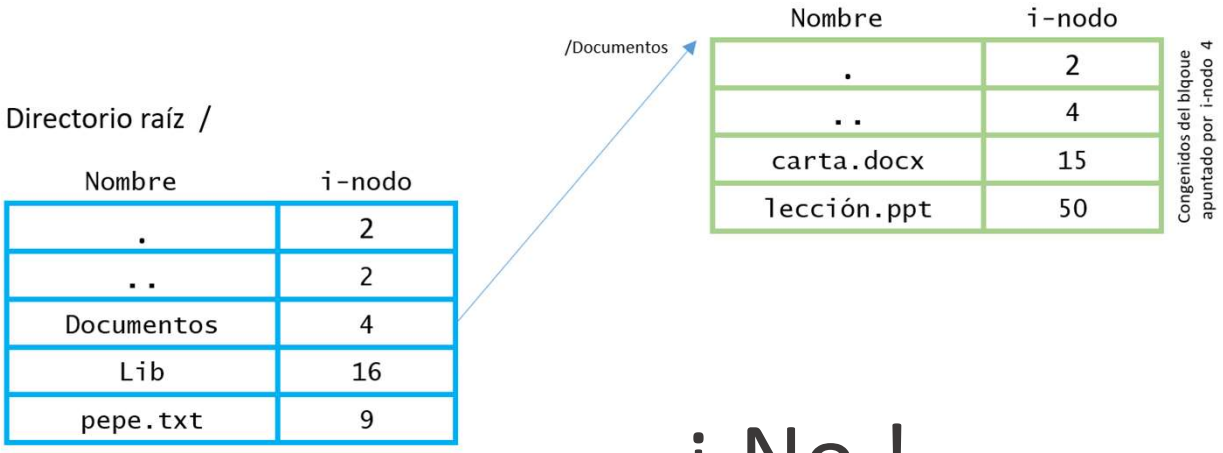
El directorio raíz está siempre en el i-nodo 2 y es padre de sí mismo

Directorio



Entonces, ¿el contenido del subdirectorio Documentos se guarda en el i-nodo 4?

Directorio



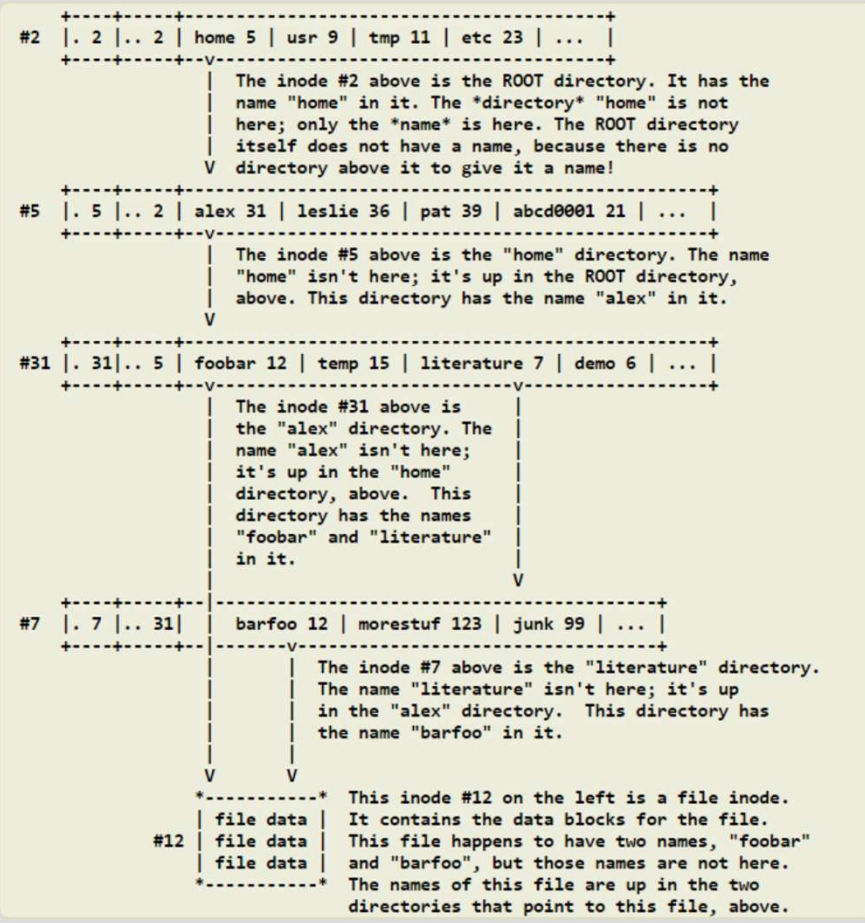
El i-nodo es una estructura que guarda información sobre los bloques que ocupa un fichero. Los contenidos del fichero están en esos bloques no en el i-nodo. Los directorios y subdirectorios son ficheros. La información del directorio raíz no está en el i-nodo 2, sino en los bloques a los que apunta el i-nodo 2 [aun no lo hemos visto]. De la misma manera, la información del subdirectorio /Documentos no está en el i-nodo 4 sino en los bloques apuntados por el i-nodo 4.

Directorio

Ejemplo de directorio.

1. `/home/alex/foobar`
2. `/home/alex/literature/barfoo`

Follow the downward-pointing arrows:



http://teaching.idallen.com/cst8207/13w/notes/450_file_system.html

Directorio

El fichero `hola.txt` se encuentra en el directorio `/home/pepe/literatura`
Completar los campos que faltan

i-nodo

Este es el directorio raíz, su i-nodo correspondiente es el 2

2 →

2 .	2 ..	home **5**	usr **9**	tmp **11**	etc **23**

porque home apunta a 5

apunta al padre

5

5 .	2 ..	pepe **31**	pili 26	luisa **39**	juan **24**

apunta a sí mismo y es hijo de 5

31

.31	5 ..	correo	UTAD **105**	literatura **7**

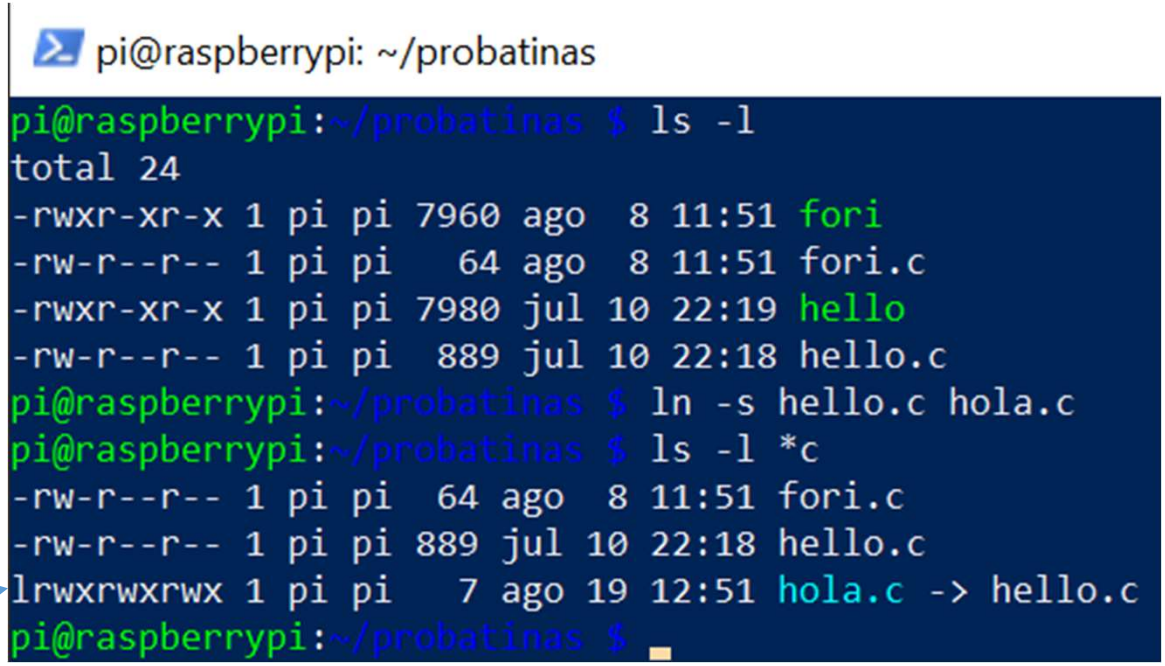
literatura apunta 7, 7 es hijo de 31

7

7 .	31 ..	MioCid **51**	hola.txt **12**

Enlaces

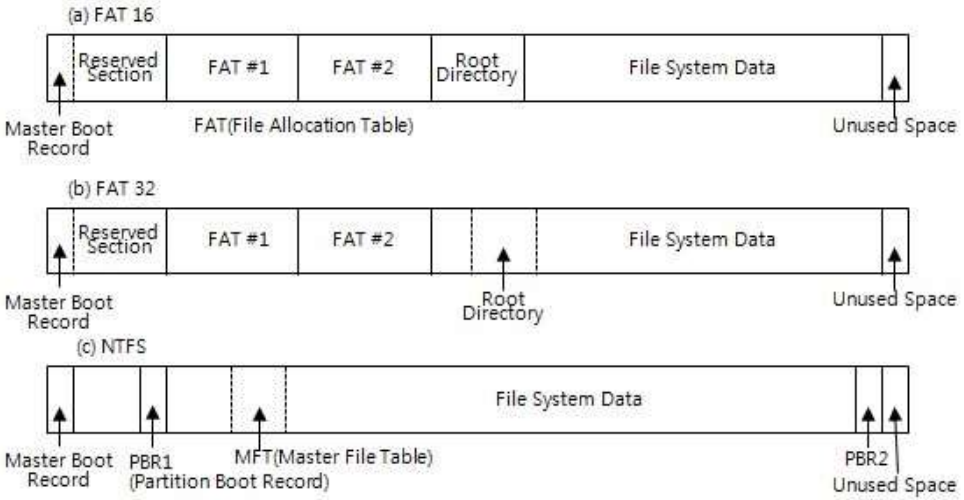
En Unix existe la posibilidad de crear enlaces o alias para un mismo fichero. El **enlace simbólico** (*soft link*) crea una nueva entrada en el directorio hacia el fichero apuntado. Si el link se borra, el fichero original no se ve afectado, si el fichero original se borra el link sigue existiendo pero dará error de acceso.



Tipo de fichero: enlace

Sistema de ficheros

Un sistema de ficheros es estructura de información que permite manejar los ficheros de una unidad de almacenamiento, que en Unix se llama **partición**. Un **volumen** coincide con un dispositivo físico, dentro de un volumen pueden definirse **unidades lógicas** (en Windows, C:, D:,…) o **particiones** en Unix. El sistema de ficheros se crea al formatear la partición en cuestión. El formato depende del Sistema que se esté usando.



Sistemas de ficheros de Microsoft.

Sistema de ficheros

Centraremos la descripción en Unix. En todo volumen (dispositivo físico: disco, USB, CR-DOM) el sector 0 es el *Master Boot Record*. En la lección 2, al ver cómo arranca el sistema operativo, aprendimos que el iniciador hardware de la BIOS busca en la lista de dispositivos de arranque hasta encontrar uno que tenga el MBR activado. También vimos que el MBR tiene 512 bytes y eso se debe a que forzosamente está en el sector 0. El MBR no forma parte del sistema de ficheros del Sistema Operativo.

Al final del MBR (últimos 64 bytes) encontramos la tabla de particiones. Una partición es una fracción lógica, equivalente a una de las unidades Windows. La tabla de particiones es única por dispositivo. En un volumen (disco) puede haber varias particiones (unidades).

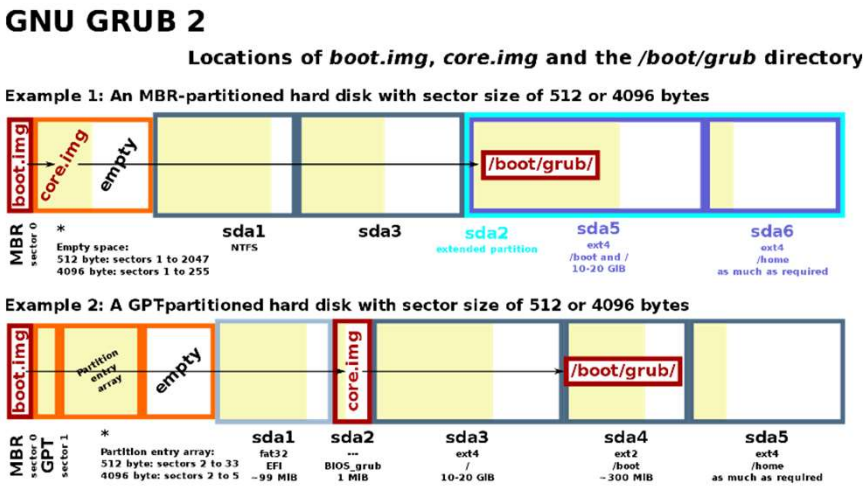
MBR tiene sus limitaciones, solo puede manejar discos de hasta 2TB y 4 particiones primarias. Para superar estos límites, se desarrolló GPT (GUID partition table). GPT permite un número casi ilimitado de particiones por volumen (128 en Windows). Las versiones de Windows de 64 bits y las distribuciones de Windows soportan arranque nativo con GPT si el equipo dispone de BIOS/UEFI, que es lo común en equipos modernos.

Sistema de ficheros

El sistema de ficheros propiamente dicho se instancia en cada partición. En todas ellas se reserva al principio el bloque de arranque, por si en ella se instala un Sistema Operativo y es la unidad desde la que debe cargarse éste.

Imaginemos un equipo con dos particiones formateadas como unidades Windows (C: y D:), con el arranque en la primera y tres particiones Linux /dev/sda1, /dev/sda2 y /dev/sda3 con el arranque también en la primera. Un cargador como GRUB dará la opción de arrancar Windows (desde C) o Linux, desde /dev/sda1/.

Cultura general, no entra como pregunta de examen, pero para que veas lo compleja que es la realidad. ¿Donde reside el cargador GRUB2?



Sistema de ficheros

El sector 0 de cada unidad se reserva para el Master Boot Record (lo vimos en el tema 2)

Tabla de particiones del disco físico

Partición (unidad lógica en Windows)

Reservado para el arranque del SSOO si está instalado en esta partición

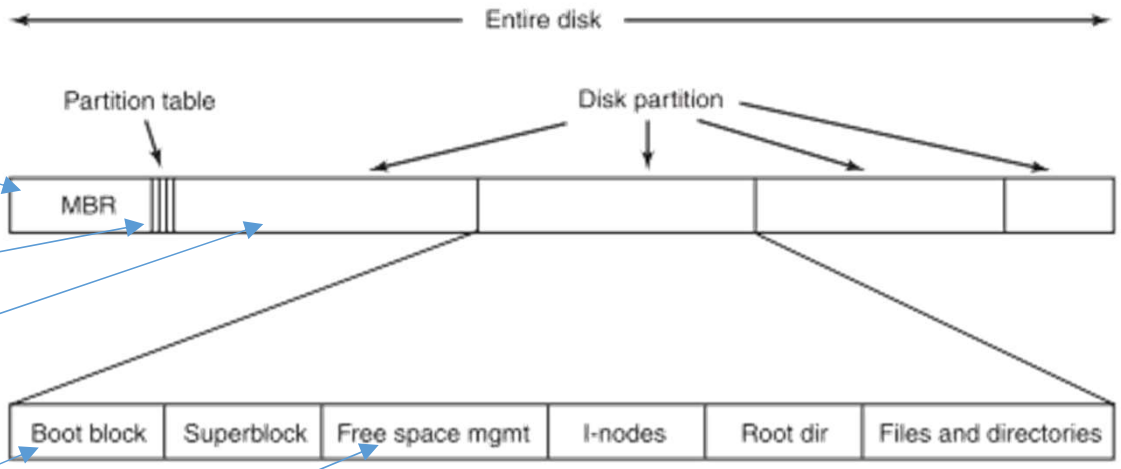


Figure 4-9. A possible file-system layout.

bitmaps

Fuente: “Understanding the Linux Kernel”, 3rd edition, Bovet & Cesati

Sistema de ficheros

El **superbloque** contiene información sobre la organización de la partición: Tamaño de la agrupación, tamaño del superbloque, bitmaps y zona de i-nodos, número de agrupaciones, número de i-nodos, número del primer i-nodo (el 2 en los sistemas actuales).

```
struct ext2_super_block {
   u32  s_inodes_count;          /* Inodes count */
   u32  s_blocks_count;          /* Blocks count */
   u32  s_r_blocks_count;        /* Reserved blocks count */
   u32  s_free_blocks_count;     /* Free blocks count */
   u32  s_free_inodes_count;     /* Free inodes count */
   u32  s_first_data_block;      /* First Data Block */
   u32  s_log_block_size;        /* Block size */
   u32  s_log_cluster_size;      /* Allocation cluster size */
   u32  s_blocks_per_group;      /* # Blocks per group */
   u32  s_clusters_per_group;    /* # Fragments per group */
   u32  s_inodes_per_group;      /* # Inodes per group */
   u32  s_mtime;                 /* Mount time */
   u32  s_wtime;                 /* Write time */
   u16  s_mnt_count;             /* Mount count */
   s16  s_max_mnt_count;         /* Maximal mount count */
   u16  s_magic;                 /* Magic signature */
   // more non-essential fields
   u16  s_inode_size;            /* size of inode structure */
}
```

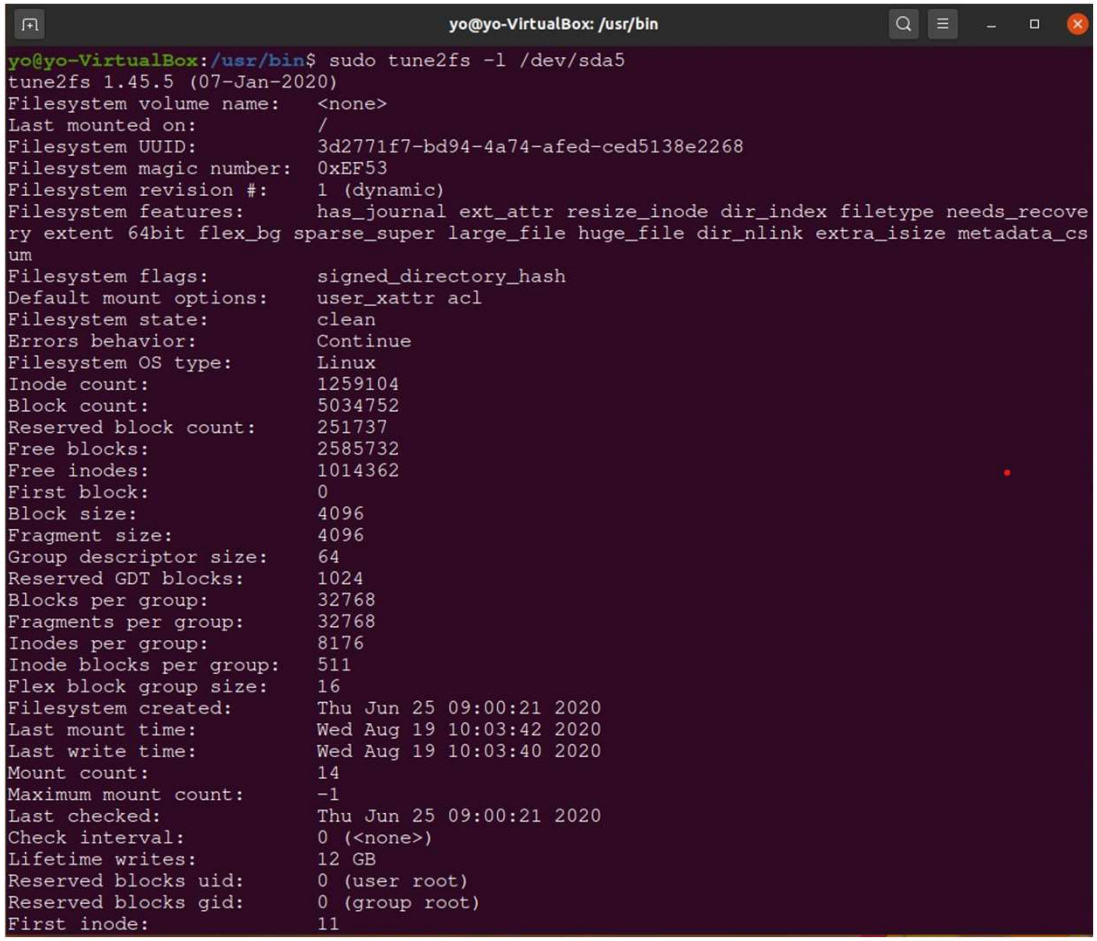
superbloque en ext2

Sistema de ficheros

Hay dos **bitmaps**, uno para i-nodos y otro para agrupaciones. El bitmap de agrupaciones contiene un bit por cada bloque para saber si está ocupado o no. En un disco de 1 TB, con bloques de 4 kB, hay $2^{40}/2^{12} = 2^{28}$ bloques. Necesitamos 2^{28} bits en el bitmap de agrupaciones. Cada bloque de información son 4 kB, que almacenan 2^{12} x 8 = 2^{15} bits. Por tanto, hay que reservar $2^{28}/2^{15} = 2^{13} = 8192$ bloques de la partición para el bitmap de agrupaciones. Esto lo calcula el comando de formateo de la partición. Puede parecer una cifra alta pero es un bit en el bitmap por cada 2^{15} bits de información útil en el bloque de un fichero. El número de **i-nodos** dependerá de la versión, pero siempre va a ser bastante menor que el de agrupaciones y su bitmap ocupará menos espacio.

Los i-nodos ocupan un espacio fijo en el sistema de ficheros, reservado en el momento del formateo. Los directorios, por el contrario, no se sabe de antemano cuánto espacio útil ocuparán, pero no es necesario. Son ficheros como otros cualesquiera, se ubican en la última zona del sistema, que es mucho mayor que todas las anteriores.

Sistema de ficheros



Con el comando `tune2fs -l` podemos ver una gran cantidad de información del sistema de ficheros

i-nodo

El **i-nodo** es la estructura más importante para la gestión de ficheros en Unix. Ha ido evolucionando, usaremos una versión sencilla la ext2 para entender su estructura.

```
struct ext2_inode {
  u16   i_mode;          // 16 bits = |tttt|ugs|rwx|rwx|rwx|
  u16   i_uid;           // owner uid
  u32   i_size;          // file size in bytes
  u32   i_atime;         // time fields in seconds
  u32   i_ctime;         // since 00:00:00,1-1-1970
  u32   i_mtime;
  u32   i_dtime;
  u16   i_gid;           // group ID
  u16   i_links_count;   // hard-link count
  u32   i_blocks;        // number of 512-byte sectors
  u32   i_flags;         // IGNORE
  u32   i_reserved1;     // IGNORE
  u32   i_block[15];     // See details below
  u32   i_pad[7];        // for inode size = 128 bytes
}
```

Punteros a bloque. Aparecerán en vuestras pesadillas

i-nodo

El **i-nodo** contiene la lista de los bloques ocupados por un fichero, además de algunos metadatos sobre permisos de acceso y fechas. Como ya hemos indicado antes, el i-nodo NO contiene el nombre del fichero, ese dato está en el directorio. Tampoco contiene su propio identificativo. ¿Cómo sabe entonces el Sistema Operativo de qué i-nodo se trata? Por la posición en el sistema de ficheros, los i-nodos se numeran consecutivamente y tienen un tamaño fijo.

Vamos ahora con la parte más complicada de entender la primera vez que se estudia este tema, la lista de bloques y los punteros indirectos.

En la figura anterior, vemos que hay doce entradas para guardar números de agrupación/bloque. Obviamente, podremos guardar hasta 12 números y como cada bloque tiene 4kB (o el tamaño fijo que se haya dado en el momento de formatear), el fichero más grande posible tendría:

$$12 \times 4\ kB = 48\ kB$$

Es un tamaño ridículo para un fichero actual, pero cuando se creó Unix 48 kB eran una cantidad importante. No obstante, con ese tamaño máximo no se puede funcionar. ¿Cuál es la solución?

i-nodo

Había dos posibles estrategias. La primera aumentar el tamaño del i-nodo y convertirlo en una lista variable, pero eso habría creado graves problemas de compatibilidad hacia atrás y de rendimiento.

La segunda es la indirección, una solución más elegante. Añadimos un nuevo campo al i-nodo, un "puntero" que contiene el número de bloque en el que se van a guardar los números de bloque adicionales. Escribo "puntero" entre comillas para no confundirlo con un puntero de memoria. La idea es la misma, pero el contenido no es una dirección de memoria como en los punteros C con los que estás familiarizado, sino un número de bloque.

Ahora aumentamos la capacidad de la lista, 12 números en el propio i-nodo más los que quepan en un bloque. ¿Cuántos? Como los números de bloque en ext2 son de 4 bytes, en los 4kB de un bloque caben 1024 números adicionales, así que subimos la cuenta total a 1036.

Tamaño máximo = (12+1024) x 4 kB, aproximadamente 4 MB (12 << 1024)

Tampoco parece la solución definitiva… pero ¿y si añadimos un campo que contiene el número de bloque en el que se almacenan los números de bloque que contienen a su vez números de bloque ocupados por el fichero? Ha nacido el puntero indirecto doble. Con eso llegamos a:

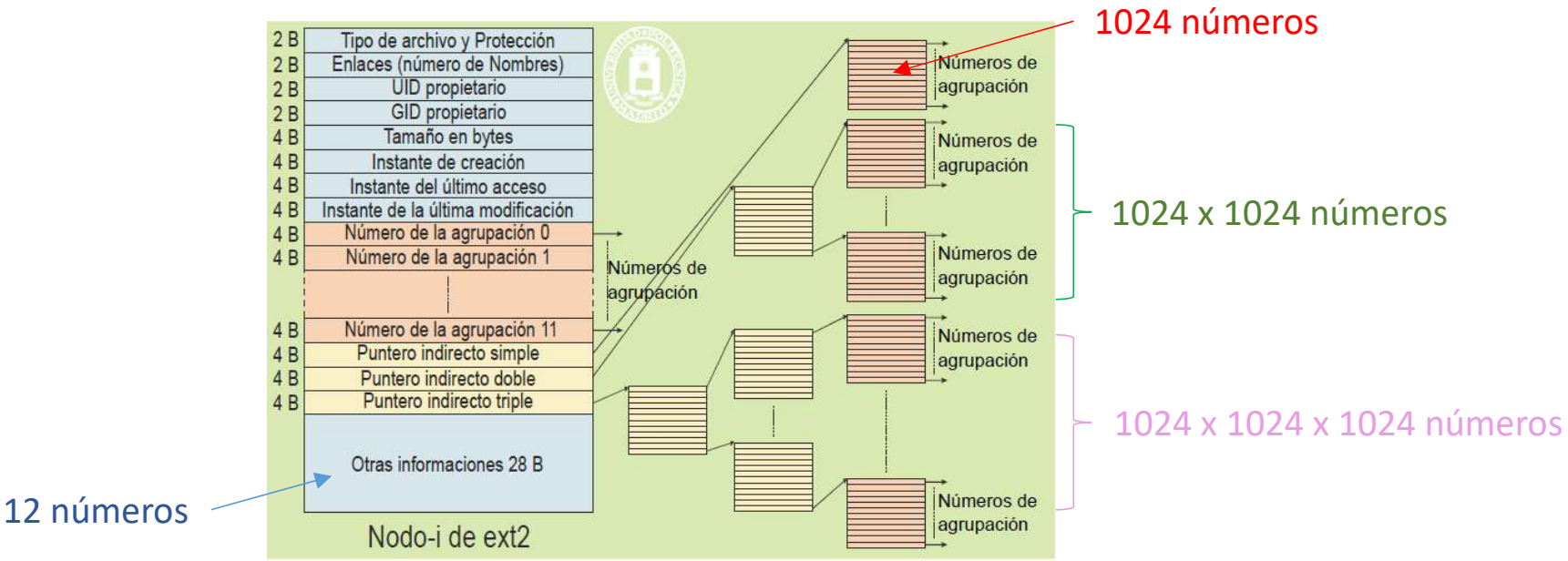
$(12 + 1024 + 1024^2)$ x 4k B, aproximadamente 4GB

i-nodo

Con 4GB podemos guardar una imagen del sistema operativo o una película en alta definición, pero todavía no es suficiente. Solución: Puntero indirecto triple. Capacidad máxima:

$$(12 + 1024 + 1024^2 + 1024^3) \times 4\ kB, \text{ aprox } 4\ TB$$

Este es el máximo teórico, en la realidad hay otros límites HW que dejaban ese tamaño en 2TB



Anasagasti, Costoya: “Sistemas Operativos”, UPM, 2016, p. 225

i-nodo

El sistema de archivos `ext` nació con Linux, `ext2` es la versión Linux de 1992, en 2001 apareció `ext3`. La estructura es muy parecida a la de `ext2` e incluye funciones adicionales de journaling que se describen más adelante en esta versión.

En 2006 se publicó `ext4` para poder manejar dispositivos de más de 4 TB. El número de bloque es de 48 bits, con lo que pueden manejarse volúmenes de 2^{48} bloques x 4 kB = 2^{60} B = 1EB (ExaByte). El tamaño máximo de fichero es de 16 TB. Una característica de `ext4` es que además de asignar bloques individuales, puede asignar rangos consecutivos de bloques, llamados ***extents***.

Sr.No.	Features	Ext3	Ext4
1	File system Limit	16TB	1EB
2	File Limit	2TB	16TB
3	Default Inode Size	128 bytes	256 bytes
4	Block Mapping	Indirect	Extent
5	Time Stamp	Second	Nanosecond
6	Sub directory limit	32768	unlimited
7	Preallocation	In core reservation	For ExtentFile
8	Defragmention	No	Yes
9	Directory Indexing	Diabled	Enabled
10	Delayed Allocation	No	Yes
11	Multiple Block Allocation	Basic	Advanced

http://linuxonmo.blogspot.com/2017/12/difference-between-ext3-and-ext4.html

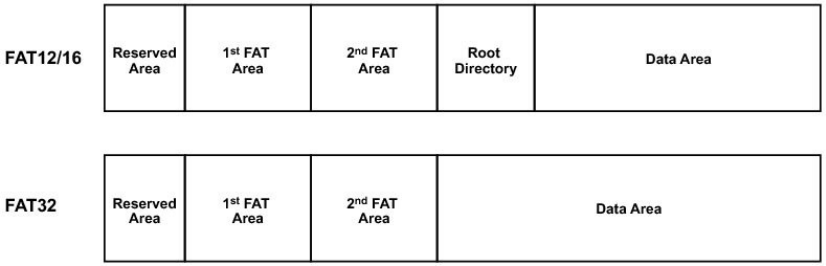
En 2009 apareció btrfs (B-tree File System) que incorpora numerosas mejoras respecto a los formatos ext (tamaño máximo, tolerancia, soporte nativo a SSD)

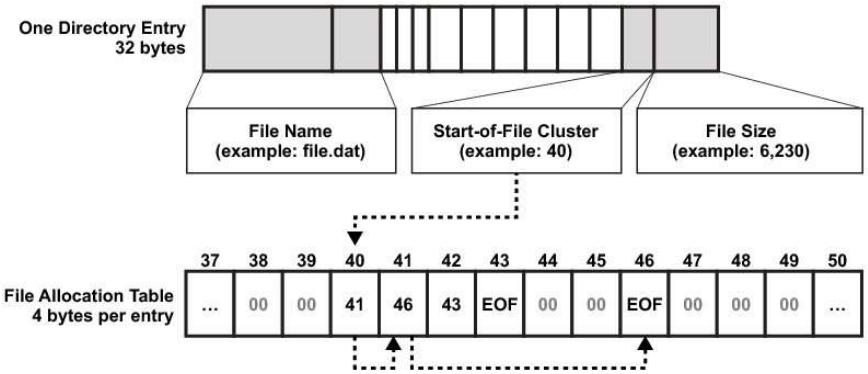
btrfs Wiki (kernel.org)

FAT

File Allocation Table (FAT) es el sistema de registro de los bloques en MS-DOS. Dada su simplicidad, se sigue usando en dispositivos como memorias USB y tanto Windows como Unix pueden leer unidades con FAT. Ha habido varias versiones, la última es FAT32.

En lugar de bitmap, en FAT se usa una lista enlazada de agrupaciones (bloques) para llevar la contabilidad de bloques ocupados y libres. FAT es un vector en el que cada entrada corresponde a un bloque. Si el bloque está libre se marca con el valor 0. Si está ocupado contiene el número del siguiente bloque ocupado por ese fichero o el número -1 (EOF) si es el último de la secuencia. En el directorio, además de los metadatos del fichero, se guarda el número del primer bloque que sirve de índice para recorrer la FAT.





Sistema de ficheros en xv6


```
struct superblock {
  uint size;         // Size of file system image (blocks)
  uint nblocks;      // Number of data blocks
  uint ninodes;      // Number of inodes.
  uint nlog;         // Number of log blocks
};
```

```
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEV only)
  short minor;          // Minor device number (T_DEV only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+1];   // Data block addresses
};
```

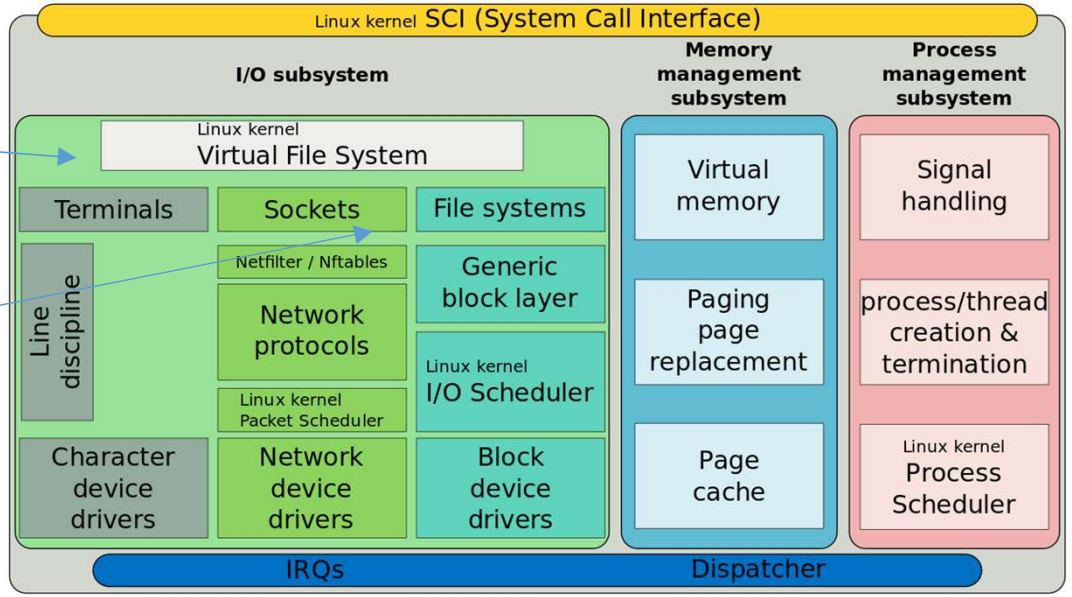
En xv6 se usa un tamaño de bloque igual al del sector de disco (512 bytes)

Sistema de ficheros

En una misma máquina pueden convivir volúmenes con distintos sistemas de ficheros. Es muy habitual tener un SSD formateado en ext4 y usar memoria USB con sistema FAT32, además puede haber unidades en red montadas por protocolo NFS. Para simplificar la gestión, en Unix se añadió una capa superior, el **Sistema Virtual de Ficheros**, que unifica el acceso. Esta capa es transparente al programador. Como se explicará en la lección 6, también permite acceder a otros recursos (pipes, memoria compartida…) con la abstracción fichero.

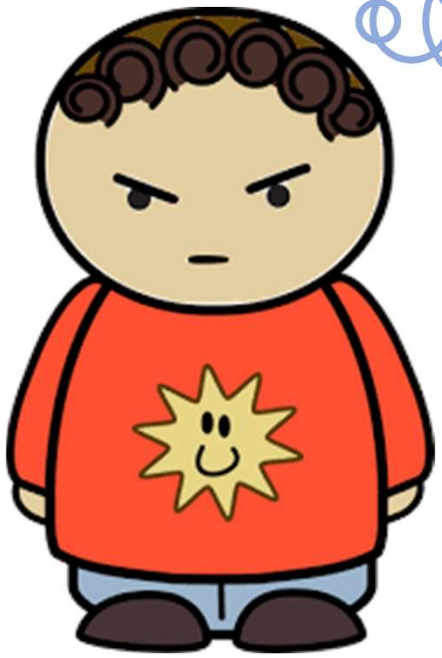
Sistema virtual de ficheros

Sistema(s) de ficheros



El sistema virtual de ficheros permite definir particiones lógicas repartidas entre varios volúmenes, pero solo sucede a este nivel

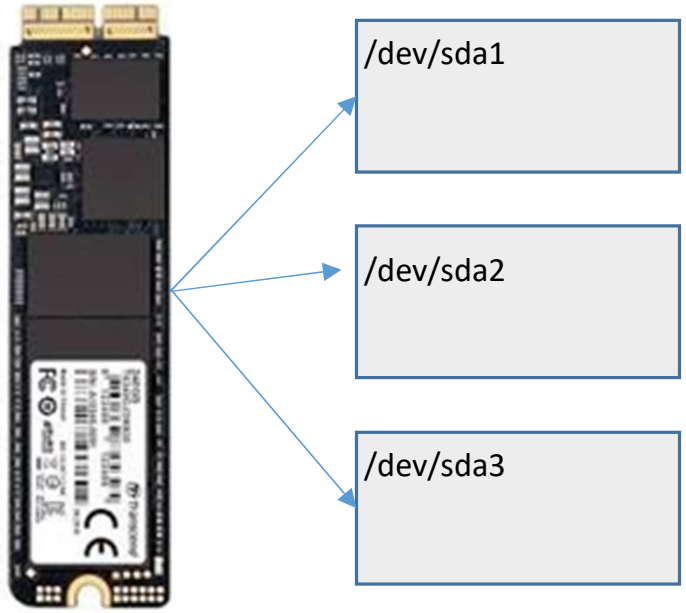
Sistema de ficheros



A estas alturas necesito un resumen, ya no sé si el bitmap se guarda en un volumen de una partición del bloque del i-nodo y con esto se construye el directorio

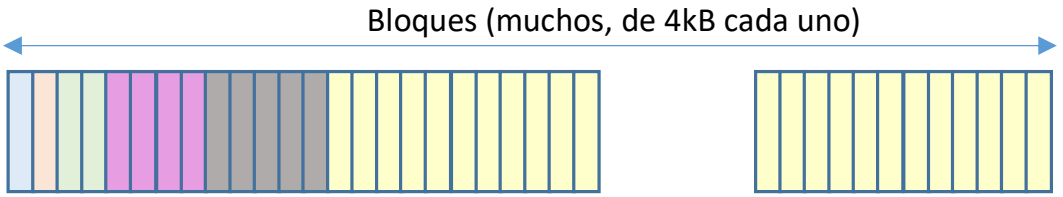
Sistema de ficheros (RESUMEN)

Sistema de ficheros de una de las particiones



Volumen (disco SSD)

Particiones. Hay una tabla al inicio del volumen que informa sobre las particiones que contiene. Cada una tiene su propio sistema de ficheros.



Bloque
Superbloque
Bitmap de inodos
Bitmap de agrupaciones
i-nodos

La información de un fichero ocupa varios bloques, como mínimo 1, pero en general muchos. La lista de los bloques que ocupa un fichero concreto está en su i-nodo. Cada fichero tiene una entrada en su directorio correspondiente que es un fichero especial. El bitmap de agrupaciones lleva la cuenta de TODOS los bloques, y sabe si están ocupados o no, pero no a qué fichero corresponden, eso lo sabe el i-nodo. Tiene que haber un bit por cada bloque. El bitmap de i-nodos, hace lo mismo para saber cuantos i-nodos hay libres. El nº inodos << nº bloques. En Unix bloque == agrupación

Servidor de ficheros

El servidor de ficheros es la parte del Sistema Operativo que ofrece al programador la abstracción **FILE**. El usuario no entiende de ext4, NTFS o HFS (el sistema propietario de Apple), solo percibe identificadores. Un identificador es un número entero positivo que retorna la operación de apertura y con el que el proceso se relaciona con el servidor de ficheros. Los identificadores abiertos por un proceso se guardan como un vector en su BCP. En xv6 es un vector.

```
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

Tamaño
Puntero a la tabla de páginas
Pila del kernel
Estado
pid
Puntero al BCP del padre
Estado del procesador
Contexto de la CPU en la pila del kernel
Vector de ficheros abiertos
Directorio de ejecución
Nombre del ejecutable

BCP de xv6

Servidor de ficheros

Cada entrada de ese vector mantiene un índice a una fila de la tabla única de ficheros abiertos que mantiene el Servidor de Ficheros. Esta tabla apunta al i-nodo del fichero.

Tabla del BCP del proceso

Identificador que se recibe al hacer fopen()

Tabla de ficheros abiertos del sistema

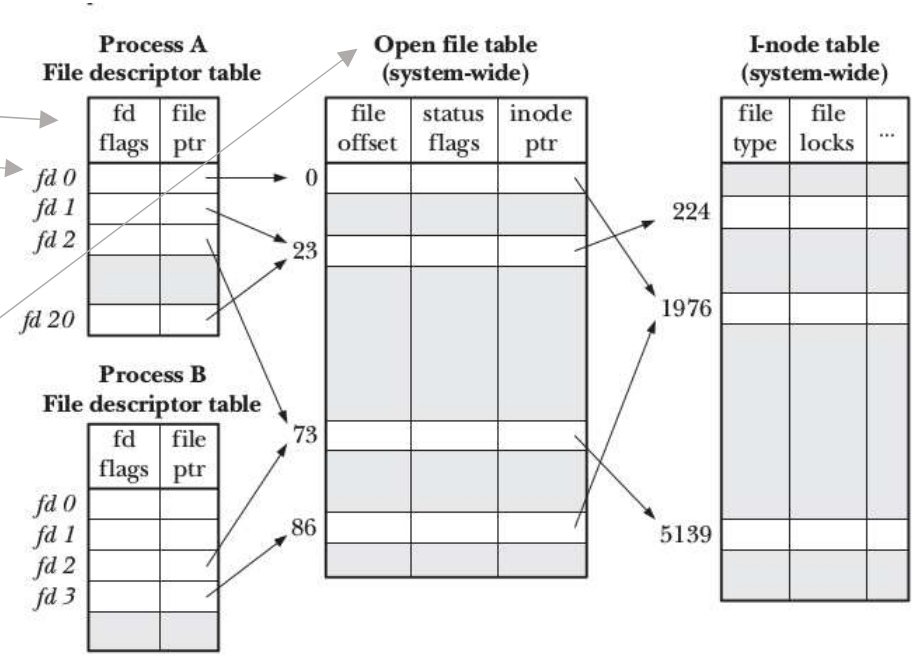


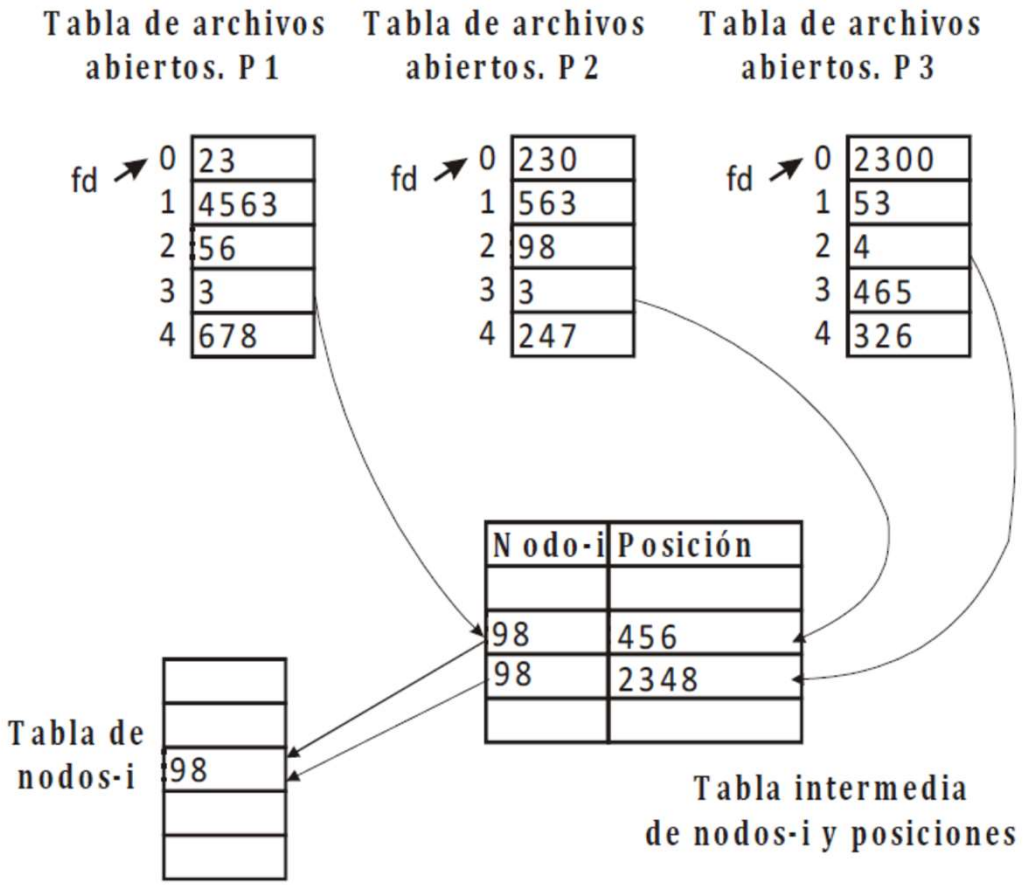
Figure 5-2: Relationship between file descriptors, open file descriptions, and i-nodes

The Linux Programming Interface

Servidor de ficheros

Todos los procesos reservan los identificadors 0, 1 y 2 para `stdin, stdout y stderr`

Puede haber varios procesos apuntando a un mismo fichero, pero sus identificadores locales son diferentes.



Servidor de ficheros

Apertura de un fichero

Para abrir un fichero, el Sistema Operativo recorre el directorio hasta localizar el nombre encontrado. Si no existe devuelve el valor `NULL`.
A continuación comprueba si el usuario tiene permisos de acceso con la información contenida en el i-nodo. Si es así, el i-nodo se instancia en la tabla de i-nodos kernel, se añade una entrada en la tabla global de ficheros y se escribe el número de esta entrada en el BCP del proceso que invocó el `fopen()`.

Creación de un fichero

Para crear un fichero, primero se comprueban los permisos del usuario en el directorio solicitado. ¿Cómo?, consultando la información del i-nodo correspondiente al directorio. Si dispone de ellos se busca un i-nodo libre en el bitmap de i-nodos, se instancia en memoria y se devuelve el valor del identificativo en la tabla intermedia al proceso llamante.

Servidor de ficheros

Lectura de un fichero

El servicio de lectura requiere un descriptor de fichero abierto, el número de bytes (N) que se van a leer y la dirección en que se almacenarán. Si todo es correcto el puntero del fichero se incrementa en N.

Escritura de un fichero

Este servicio permite escribir M bytes, devuelve el número de bytes realmente escrito. El servicio se tiene que ocupar de buscar agrupaciones libres (con el bitmap de agrupaciones) para poder ampliar el fichero actual, si es el caso.

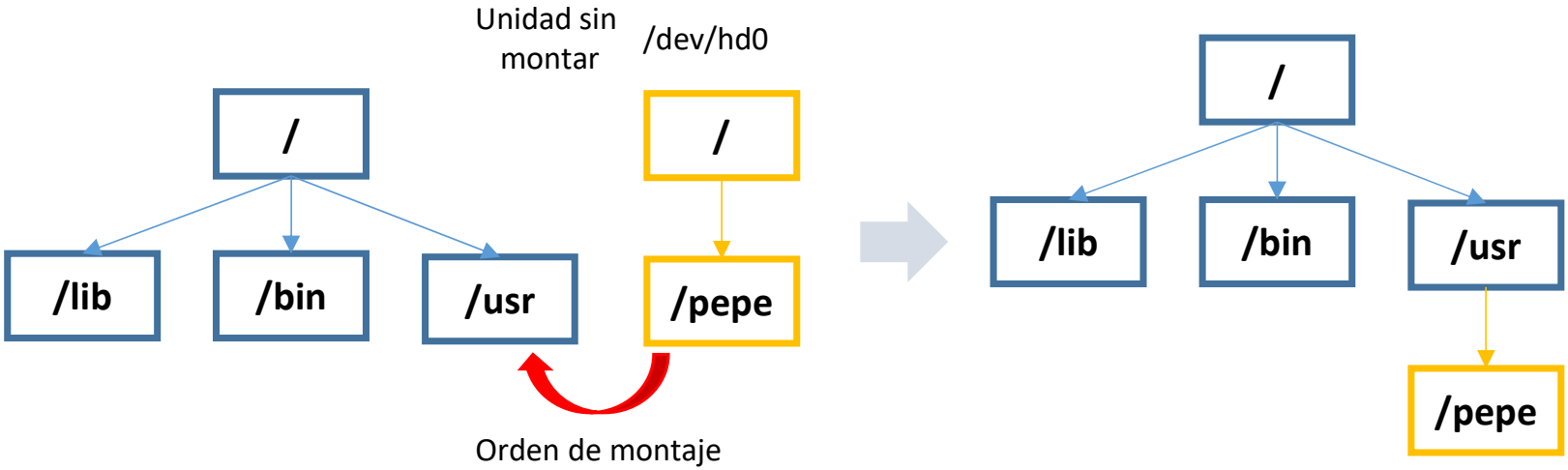
Cierre de un fichero

Cuando un proceso pide cerrar un fichero, o simplemente termina y el SO se ocupa de invocar el cierre, la entrada de la tabla intermedia se marca como libre. Si no hay otros procesos usándolo se libera también la información en la tabla de i-nodos.

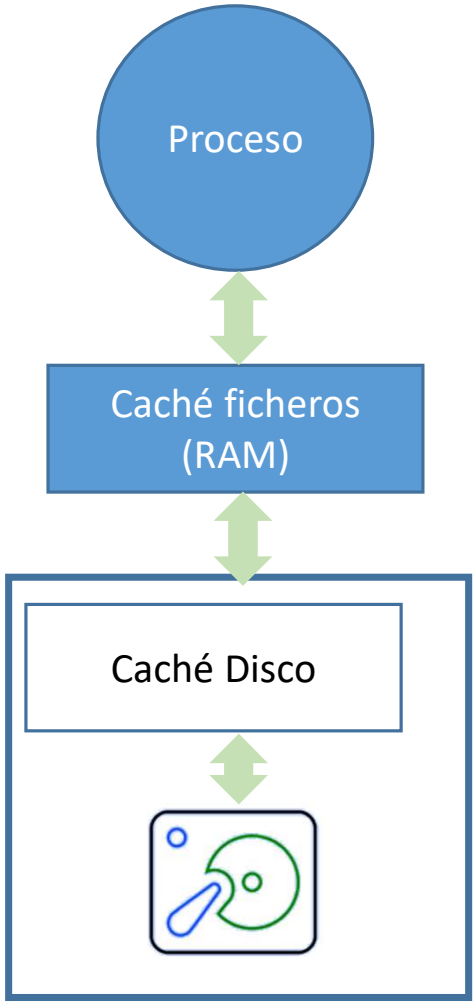
Montaje

En los sistemas Unix la jerarquía de ficheros es única, todos los directorios descienden en último término del raíz /. Esto no ocurre así en Windows donde cada unidad lógica, equivalente a una partición Unix es autocontenida.

El mecanismo de Unix para poder ver todos los ficheros en un solo árbol se denomina **montaje**. Este servicio permite proyectar la estructura de un servicio de ficheros en el directorio de otro.



Caché de ficheros



Para aumentar el rendimiento se usan DOS cachés. Una es del propio disco y la maneja su controlador, es opaca al Sistema Operativo.

La caché de ficheros es una zona de RAM que sí gestiona el Sistema Operativo de forma directa y utiliza la misma propiedad de proximidad referencial que la caché de memoria o que la memoria virtual. Se trata de reducir al mínimo los accesos a disco físico.

Caché de ficheros

La operación más costosa en tiempo es la escritura en disco. Para evitarlo, se aplican diferentes políticas de sincronización:

- **Write through** (inmediata): Un bloque se escribe en cuanto se modifica, equivale que no haya caché.

- **Write back** (escritura diferida): Los datos solo se escriben desde la caché a disco cuando ésta se ha llenado y hay que reemplazarlos. Si durante la ejecución hay varios cambios en un mismo bloque solo se reflejan en la caché, por lo que podrían perderse si se pierde la alimentación.

- **Delayed write** (escritura con retardo): La sincronización se produce de forma periódica, lo que supone un compromiso entre los dos modos anteriores y reduce el riesgo de pérdida de información.

- **Write on Close** (escritura al cierre): En el cierre de un fichero siempre se produce una sincronización, con independencia de la política usada. Con write on close, la diferencia es que solo se escribe en disco al cierre, no antes.

Journaling

Uno de los puntos débiles de los sistemas Unix primitivos era su fragilidad ante una parada desordenada por pérdida de alimentación u otra causa como un kernel panic. Hay varios niveles de caché e incluso sin ellos, la transferencia se hace por bloques completos (4kB) que son 16 sectores de disco.

Si por cualquier circunstancia una escritura no llegaba a completarse el fichero quedaba inconsistente. Si tenemos la mala suerte de que esas zonas corrompidas correspondan a i-nodos, a los bitmaps o a un directorio, ya no es solo que un fichero queda mal sino que podemos perder gran parte del sistema de ficheros.

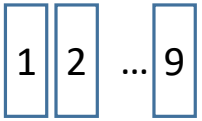
Para evitar este problema se añadieron los sistemas de *journaling*, que podemos traducir como diario de transacciones. Tenemos la idea de lo que es una transacción en base de datos, un conjunto de operaciones que deben completarse de forma correcta, si no es así, el sistema es capaz de deshacerlas (roll back) revirtiendo cada operación individual. La idea es similar para la escritura en disco. Es necesario anotar en el *journal* cada operación para poder hacer ese *roll back* y eso tiene coste en términos de espacio y rendimiento.

Journaling

- Cada vez que se llama a un servicio de disco, los cambios en sus metadatos se registran en el *journal*. Las entradas correspondientes a una mismo servicio constituyen una transacción.
- Cuando una operación individual se completa, por ejemplo escribir un bloque, se anota el resultado.
- Si todas las operaciones de la transacción se cursan con éxito, los datos de esa transacción se borran del *journal*.
- Si el sistema sufre un *crash*, el sistema puede saber en qué punto se suspendieron las transacciones que estaban abiertas y puede revertirlas. Esto ocurre en el arranque del sistema y es lo que explica por qué cuando apagamos nuestro equipo físicamente, el sistema operativo tarda tanto en cargar en el siguiente arranque.

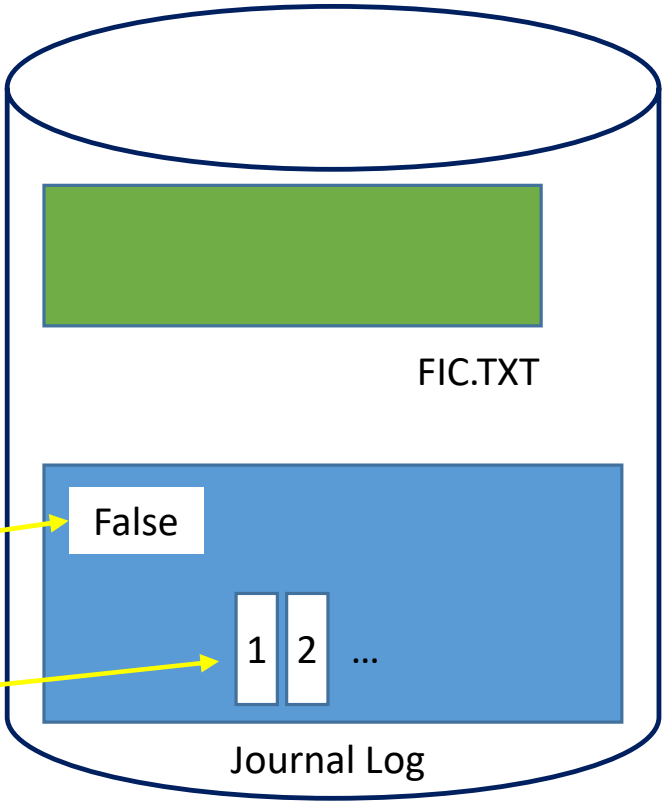
Journaling

La operación completa (transacción) consiste en escribir 9 bloques (no necesariamente consecutivos) en el fichero FIC.TXT



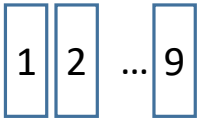
1. Se crea una entrada con el valor de la transacción a False

2. Se van copiando los bloques uno a uno



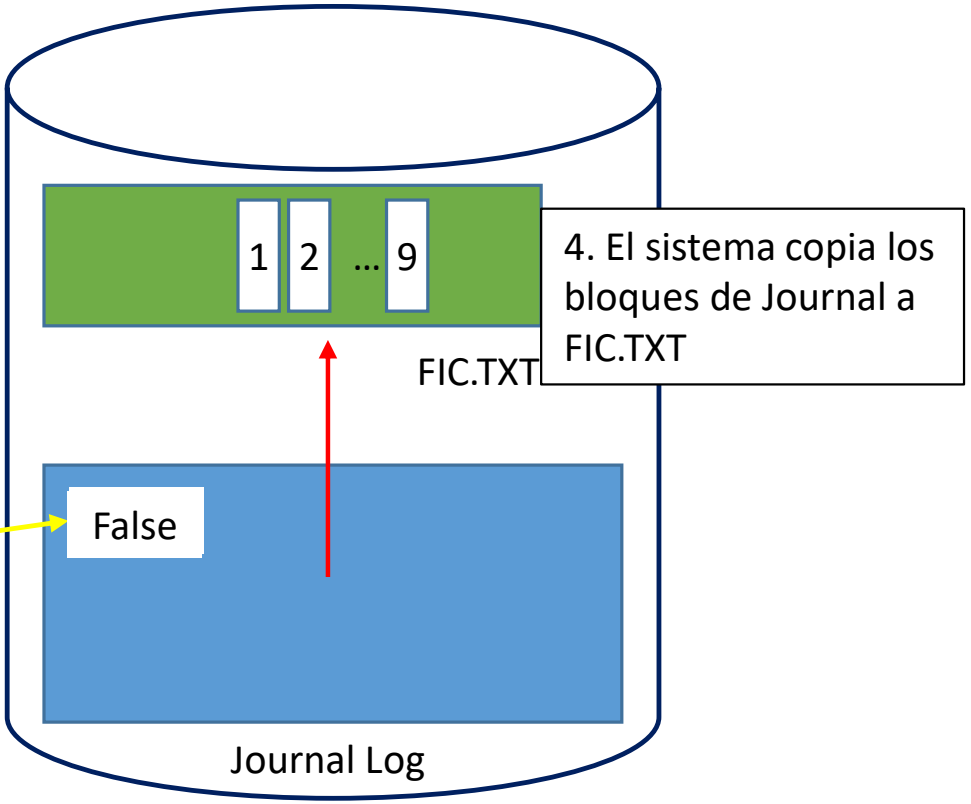
Journaling

La operación completa (transacción) consiste en escribir 9 bloques (no necesariamente consecutivos) en el fichero FIC.TXT



3. Si la transacción termina bien se cambia a True

5. El sistema pone a False el estado de la transacción y borra los bloques del Journal. Al terminar, borra el flag. Fin de transacción



Journaling

- Si el sistema sufre un *crash* durante la copia inicial, el flag está a False. Al rearrancar se elimina la transacción incompleta del log, incluyendo los bloques que se hubiesen podido escribir.

- Si durante el arranque el Journal se encuentra una transacción completa (True) en el Log, copiará todos los bloques desde el Log al fichero.

- Si el fallo se produce durante el borrado de los ficheros del Log, en el arranque el sistema se encuentra el flag a False. Los bloques se han copiado bien al fichero pero la transacción no se ha completado. Desde el punto de vista del gestor de Journal es la misma situación que en el primer punto, una transacción incompleta con bloques en el Journal. Borra todo.