Resumen File System

Tomás Felipe Melli

Noviembre 2024

Índice

1 Qué consideraciones hay que tener cuando muchos procesos quieren acceder a distintos bloques de disco ?

Este *ordenamiento lógico de la información dentro de un disco* es el **File System** o sistema de archivos. Gracias a ellos, podemos organizar la información del usuario y del SO dentro de un dispositivo de almacenamiento.

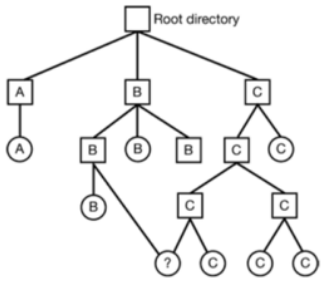
1.1 Qué es entonces un File System ?

Es un **módulo dentro del kernel** encargado de organizar la información en disco. Según el SO, puede soportar uno o varios sistemas de archivos.

- **DOS** : sólo FAT (File Allocation Table)
- **Windows** : FAT, FAT32, NTFS (New Technology File System), ...
- **UNIX** : con módulos dinámicos de kernel se puede lograr que se soporte cualquiera.
- Otros FS (File System) populares : UFS, FFS, EXT2, EXT3, EXT4, ISO-9660 (Para grabar un CD), Existen algunos *Distribuidos* como NFS, DFS, ...

2 Qué responsabilidades tiene ?

1. **Organización Interna** : estructura interna del archivo (windows y unix secuencia de bytes). La responsabilidad es del usuario.
2. **Organización Externa** : cómo se ordenan ? Limitaciones ? (pueden tener el mismo nombre ?). Aparece el concepto de **directorio** donde se conforma **una estructura jerárquica con forma de árbol**.



3. **LINK**, enlace o acceso directo : hacer referencia a otro lugar desde uno distinto, *alias*. Hay que tener cuidado con esto porque si generamos una herramienta para recorrer este árbol y se hace referencia a un antecesor **(ciclo)** podría no terminar el programa.
4. **Nombramiento** : determina cómo serán nombrados, caracteres de separación de directorio, si tienen extensión, restricciones de longitud o de uso de caracteres, distinción de mayúsculas, **Punto de Montaje**. Esto último significa **en qué parte del árbol de directorio voy a ver un FS**. Cuando tenemos más de una partición o unidades de almacenamiento, tenemos que definir cuál de todas estamos usando y avisarle al SO. **Se hace accesible el disco**. Este término viene del montaje de la cinta en el carretel (historia).

3 Qué es un Archivo ?

Un **archivo** es **una secuencia de bytes sin estructura** (a nivel del SO ya que no sabe qué es). El SO los identifica con su nombre. La *extensión* ayuda a saber qué contiene (.txt, .cpp, .tex), sin embargo, *no determina qué hay dentro.*
Existen varios **tipos definidos** :

- **D** : directorio
- **-** : arhivo regular
- **l** : enlace simbólico
- entre otros tantos ...

Recordemos que en un disco la *unidad más pequeña es un* **sector** *de 512 Bytes.* Normalmente, los FS se dividen en bloques MUCHO más grandes, 4 KB. Esto nos permite organizar la memoria de varias maneras :

- **Bloques Contiguos** : como vimos en Administración de Memoria, esto es copado para leer ya quee es rápido, pero aparecen problemas cuando el archivo crece y no hay espacio, o se produce *fragmentación.* Si bien es ineficiente para *lecto-escritura*, anda muy bien para usos de sólo lectura, sin modificar, como un CD.
  Ejemplo : FS ISO 9660
- **Lista Enlazada** : soluciona el problema de un archivo que crece en tamaño porque se le puede asignar cualquier bloque, pero si queremos hacer una lectura arbitraria (supongamos que queremos leer el bloque 4), sí o sí tiene que comenzar desde el inicio, hasta lograr capturar ese bloque, y por tanto, *se nos va en tiempo de acceso.*
- La **solución** a lo anterior es *implementar una tabla* de forma que *por cada bloque, me da la continuación del fragmento de cada archivo*

Ejemplo. El archivo A está en los bloques 1, 2, 5, 7 y 9, el archivo B en los bloques 4, 3 y 8.

Bloque	Siguiente
0	vacío
1	2
2	5
3	8
4	3
5	7
6	vacío
7	9
8	-1
9	-1

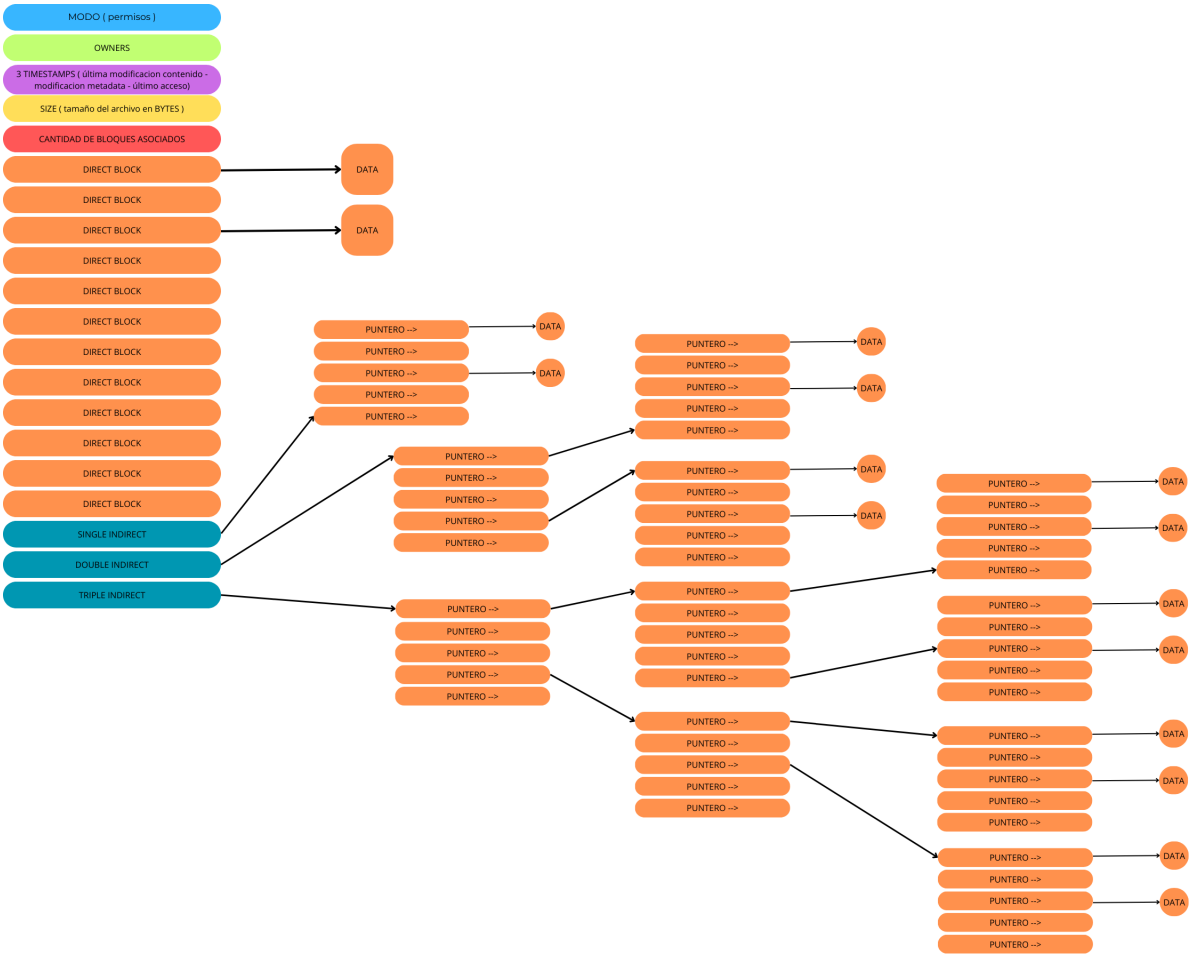
4 FAT (File Allocation Table)

Es un FS que hace uso de la tabla anterior.
Es muy conocido ya que es reconocido por muchos dispositivos, es simple.
Un **Cluster** de datos **es la mínima unidad de espacio** en disco que el fs utiliza para almacenar datos.
El tema de FAT es que **la tabla debe estar en memoria**. Por ser única, **hay contención**. Y para discos grandes, puede ser inmanejable. Supongamos que alguien patea el enchufe, como no se bajó a disco, los cambios se pierden. El tema de seguridad, es no menos, ya que **no se definen permisos**

5 UNIX crea un FS basado en Inodos

5.1 Qué es un Inodo

Un **inodo** es una estructura de datos particular que se define **por cada archivo**, es decir que cada uno, tiene un inodo asociado. La ventaja es que **permite tener en memoria sólo las tablas correspondientes a los archivos abiertos** y al tener una tabla por archivo, hay mucha menos contención. Dado que sólo están en memoria las listas correspondientes a los archivos abiertos, hay también mayor consistencia.



Si prestamos atención, en esta estructura tenemos almacenados los **permisos**, los dueños del archivo **(owner)**, **3 timestamps**, **size** en bytes, **cantidad de bloques asociados**, una serie de **bloques directos** es decir, **LBA (logical block address)** que apuntan directamente a la data que queremos guardar, un bloque de **indirección simple**, un bloque de **indirección doble** y un bloque de **indirección triple**. Esto de la indirección lo que quiere decir es que la LBA apunta a un bloque lleno de otras LBA que, en el caso de simple, apuntan estas últimas a bloques de datos, sino, la repite esa lógica de anidar LBA en otros bloques llenos de LBA.
Además, guarda los **atributos**, es decir **metadata** que incluye lo que ya vimos pero además,**bit de archivado, tipo de archivo, Flasgs, conteo de referencia**.
Cómo se maneja el espacio libre ?
Una manera es un **Bit Map Empaquetado** donde los bits en 1 indican espacio libre. Se puede implementar también una lista enlazada de bloques libres. Otra manera, es que cada nodo de la lista indique además del puntero, *cuántos libres consecutivos hay a partir de él.*
Cómo sabemos qué cantidad de LBA's podemos almacenar en un bloque ? La manera de calcular **la cantidad de LBA** que hay en cada tablita es conociendo a priori el **tamaño de bloque** y el **tamaño de las**

LBA(normalmente de 4 Bytes) entonces ...

$$\#LBAs = \frac{Tamaño\_Bloque}{Tamaño\_LBA}$$

5.2 Cómo se implementa un directorio con Inodos ?

Los directorios se representan también con inodos estructuralmente idénticos. La diferencia es que, en vez de contener datos en los bloques apuntados por las LBA's, contienen **Dir_Entries**. Ejemplo de EXT2 :

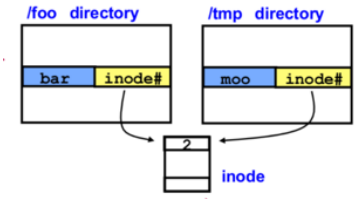
```
struct __attribute__((__packed__)) Ext2FSDirEntry {
        unsigned int inode;
        unsigned short record_length;
        unsigned char name_length;
        unsigned char file_type;
        char name[];
};
```

6 Links (Enlaces)

6.1 Hard Link

El **Hard-Link** es más rápido ya que apunta *directamente al inodo del archivo*, eso le da la capacidad de seguir accediendo a los datos, si el original se elimina. La *limitación que tiene es que es local al file system.* Se crean con

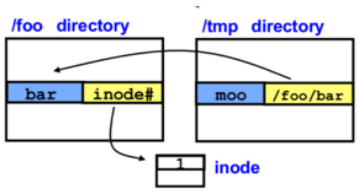
```
    ln /foo/bar /tmp/moo
```



6.2 Symbolic Link

Como respuesta a la flexibilidad (esto de que esté entre FS diferentes), aparece el **symbolic-link**. Tiene la capacidad de apuntar a cualquier archivo en el mismo FS o en otro. La desventaja es que si se elimina el archivo original, queda un huérfano. Es más lento ya que no tiene un link directo al inodo.
Supongamos dos particiones del disco formateada una con EXT4 y otra con NFTS. Si tenemos *texto.txt* en una partición y otro *texto.txt* en la otra, **no podremos hacer hard-link** ya que están en dos FS diferentes, **pero sí un symbolic-link**.
Para crearlos

```
    ln -s /foo/bar /tmp/moo
```

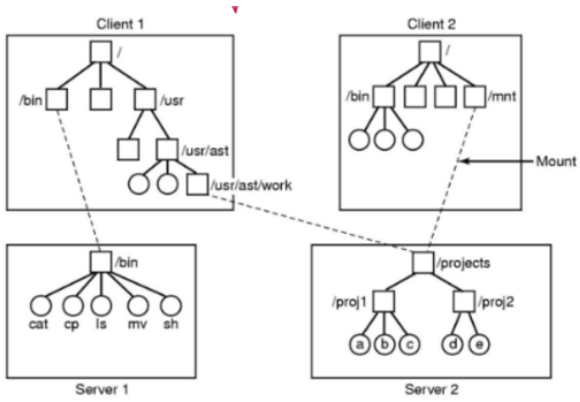


7 Consistencia de datos

La **consistencia de los datos** es vital. Ya que podría darse un problema eléctrico o algún otro imprevisto antes de que los datos se graben en el disco y se perderían los datos.
Existe una **syscall** llamada **fsync()** que *le indica al SO que queremos que las cosas se graben sí o sí* (las páginas "dirty" del cache). Sin embargo, podría darse un problema allí. Para ello, la alternativa tradicional es **un programa que restaura la consistencia del FS**, en UNIX se llama **FSCK (File System Consistency Check)** que recorre todo el disco y por cada bloque cuenta *cuántos inodos le apuntan y cuántas veces aparece referenciado en la lista de bloques libres.*
La ides es agregarle al FS un *bit que indique apagado normal*, si se levanta el sistema y este no está prendido, se corre FSCK. Esto toma **muuucho tiempo** y se debe esperar a que termine. A partir de esta demora aparece el **JOURNALING** que matiene un **log** o **journal** de las operaciones que se van a realizar en el FS antes de que sucedan. Se graba un buffer circular. Cuando se baja el cache a disco, se actualiza una marca indicando *qué cambios ya se reflejaron.* Si el buffer se llena, se baja el cache a disco. El impacto en performance es bajo ya que este registro escribe en bloques consecutivos. Cuando el sistema levanta, se aplican los cambios qe quedaron sin aplicar, y listo.

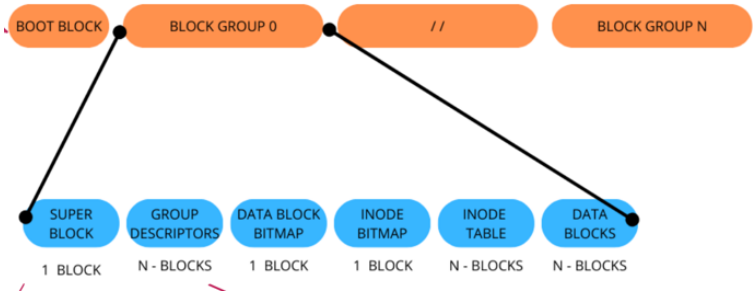
8 NFS (Network File System)

Es un **protocolo** que permite el acceso a **file systems remotos** como si fueran locales usando **RPC(Remote Procedure Call)**, un mecanismo de comunicación entre procesos que permite a un programa ejecutar procedimientos o funciones en un equipo remoto como si estuvieran en el mismo espacio de direcciones que el programa local. Todo esto es posible gracias a una **capa de abstracción (un nivel de separación entre diferentes partes de un sistema que permite a los desarrolladores y usuarios interactuar con componentes sin necesidad de conocer su funcionamiento interno ej: una API)** del SO llamada **VFS (Virtual File System)** donde cada archivo y directorio en VFS se representa mediante un inodo virtual que puede referirse a inodos reales en sistemas de archivos específicos.



9 EXT2 : Extended File System 2

Es un sistema de archivos utilizado en LINUX, pero compatible con muchos otros SO.
Tiene la siguiente estructura :


BOOT BLOCK
BLOCK GROUP 0
/ /
BLOCK GROUP N
SUPER BLOCK
GROUP DESCRIPTORS
DATA BLOCK BITMAP
INODE BITMAP
INODE TABLE
DATA BLOCKS
1 BLOCK
N - BLOCKS
1 BLOCK
1 BLOCK
N - BLOCKS
N - BLOCKS

9.1 Superbloque

El **Superbloque** contiene metadatos críticos del sistema de archivos como la cantidad de espacio libre, Si se daña o pierde la data no podría determinar qué partes de FS contienen información.

Type	Field	Description
__le32	s_inodes_count	Total number of inodes
__le32	s_blocks_count	Filesystem size in blocks
__le32	s_r_blocks_count	Number of reserved blocks
__le32	s_free_blocks_count	Free blocks counter
__le32	s_free_inodes_count	Free inodes counter
__le32	s_first_data_block	Number of first useful block (always 1)
__le32	s_log_block_size	Block size
__le32	s_log_frag_size	Fragment size
__le32	s_blocks_per_group	Number of blocks per group
__le32	s_frags_per_group	Number of fragments per group
__le32	s_inodes_per_group	Number of inodes per group
__le32	s_mtime	Time of last mount operation
__le32	s_wtime	Time of last write operation
__le16	s_mnt_count	Mount operations counter
__le16	s_max_mnt_count	Number of mount operations before check
__le16	s_magic	Magic signature
__le16	s_state	Status flag
__le16	s_errors	Behavior when detecting errors
__le16	s_minor_rev_level	Minor revision level
__le32	s_lastcheck	Time of last check
__le32	s_checkinterval	Time between checks
__le32	s_creator_os	OS where filesystem was created
__le32	s_rev_level	Revision level of the filesystem
__le16	s_def_resuid	Default UID for reserved blocks
__le16	s_def_resgid	Default user group ID for reserved blocks
__le32	s_first_ino	Number of first nonreserved inode
__le16	s_inode_size	Size of on-disk inode structure
__le16	s_block_group_nr	Block group number of this superblock
__le32	s_feature_compat	Compatible features bitmap
__le32	s_feature_incompat	Incompatible features bitmap
__le32	s_feature_ro_compat	Read-only compatible features bitmap
__u8 [16]	s_uuid	128-bit filesystem identifier
char [16]	s_volume_name	Volume name
char [64]	s_last_mounted	Pathname of last mount point
__le32	s_algorithm_usage_bitmap	Used for compression
__u8	s_prealloc_blocks	Number of blocks to preallocate
__u8	s_prealloc_dir_blocks	Number of blocks to preallocate for directories
__u16	s_padding1	Alignment to word
__u32 [204]	s_reserved	Nulls to pad out 1,024 bytes

9.2 Descriptor de Grupo

Como cada grupo puede contener varios usuarios, el **descriptor de grupo** ayuda al sistema operativo a manejar los permisos y el acceso a los recursos.

Type	Field	Description
__le32	bg_block_bitmap	Block number of block bitmap
__le32	bg_inode_bitmap	Block number of inode bitmap
__le32	bg_inode_table	Block number of first inode table block
__le16	bg_free_blocks_count	Number of free blocks in the group
__le16	bg_free_inodes_count	Number of free inodes in the group
__le16	bg_used_dirs_count	Number of directories in the group
__le16	bg_pad	Alignment to word
__le32 [3]	bg_reserved	Nulls to pad out 24 bytes

9.3 Qué más tiene un inodo en EXT2 ?

Type	Field	Description
__le16	i_mode	File type and access rights
__le16	i_uid	Owner identifier
__le32	i_size	File length in bytes
__le32	i_atime	Time of last file access
__le32	i_ctime	Time that inode last changed
__le32	i_mtime	Time that file contents last changed
__le32	i_dtime	Time of file deletion
__le16	i_gid	User group identifier
__le16	i_links_count	Hard links counter
__le32	i_blocks	Number of data blocks of the file
__le32	i_flags	File flags
union	osd1	Specific operating system information
__le32 [EXT2_N_BLOCKS]	i_block	Pointers to data blocks
__le32	i_generation	File version (used when the file is accessed by a network filesystem)
__le32	i_file_acl	File access control list
__le32	i_dir_acl	Directory access control list
__le32	i_faddr	Fragment address
union	osd2	Specific operating system information