

Sistemas de archivos

Rodolfo Baader

Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Sistemas Operativos, segundo cuatrimestre de 2024

(2) Archivos

- ¿Qué es un archivo?
- Para nosotros (y para los sistemas operativos como Unix): una secuencia de bytes, sin estructura.
- Se los identifica con un nombre.
- El nombre puede incluir una extensión que podría servir para distinguir el contenido. Por ejemplo:
 - archivo.txt: archivo con contenido de texto
 - archivo.tex: archivo con un fuente de Latex
 - archivo.c: archivo con código fuente en C.

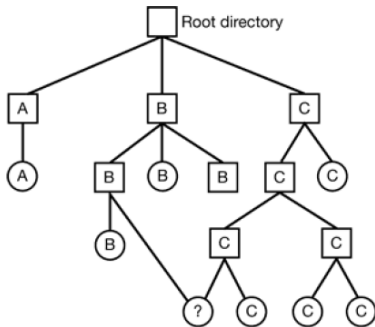
(3) Los sistemas de archivos

- Existe un módulo dentro del kernel encargado de organizar la información en disco: *sistema de archivos* o *file system*.
- Algunos SO soportan sólo uno (ejemplo: DOS sólo soporta FAT),
- ...otros más de uno (Windows soporta FAT, FAT32, NTFS, etc.),
- ... y otros como los Unix modernos suelen venir con soporte para algunos pero mediante módulos dinámicos de kernel se puede hacer que soporten casi cualquiera.
- Otros file systems populares: UFS, FFS, ext2, ext3, ext4, XFS, RaiserFS, ZFS, ISO-9660.
- Existen incluso file systems distribuidos, es decir, file systems donde los datos están distribuidos en varias máquinas en la red. Por ejemplo: NFS, DFS, SMBFS, AFS, CodaFS, etc.

(4) Responsabilidades del FS

- Una de las responsabilidades más elementales es ver cómo se organizan, de manera lógica, los archivos.
 - Interna: como se estructura la información dentro del archivo. E.g. Windows y Unix usan secuencia de bytes. La responsabilidad es del usuario.
 - Externa: cómo se ordenan los archivos. Hoy en día todos los FS soportan el concepto de directorios, lo que hace que la organización sea jerárquica, con forma de árbol.
- Casi todos, además, soportan alguna noción de *link*.
- Un link es un alias, otro nombre para el mismo archivo. Teniendo links la estructura deja de ser arbórea y se vuelve un grafo dirigido propiamente dicho, con ciclos y todo.

(5) Árbol de directorios




(6) Responsabilidades del FS (cont.)

- Además, el FS determina cómo se nombrará a los archivos.
 - Caracteres de separación de directorio.
 - Si tienen o no extensión.
 - Restricciones a la longitud y caracteres permitidos
 - Distinción o no entre mayúsculas y minúsculas.
 - Prefijado o no por el equipo donde se encuentran.
 - Punto de montaje.
- Ejemplos:
 - `/usr/local/etc/apache.conf`
 - `C:\Program Files\Antivirus\Antivirus.exe`
 - `\\SERVIDOR3\Parciales\parcial1.doc`
 - `servidor4:/ejercicios/practica3.pdf`

(7) ¿Qué es el punto de montaje?

- Imaginemos que tengo más de un disco.
- En realidad, más de una unidad de almacenamiento externo.
- Por ejemplo (histórico), una cinta.
- Monto la cinta (es decir, la coloco en el carretel).
 - ¿Cómo me refiero a ella?
 - ¿Cómo sabe el SO que ya está colocada?
- En aquel momento a esta operación se la llamaba *montaje*.
- Y para eso se inventó el comando mount, que le indicaba al SO que la cinta ya estaba colocada, y qué punto del grafo de nombres de archivos íbamos a considerar como la raíz de la cinta.
- Eso permite cosas como
`/dispositivos/cintas/cinta1/Fernandez/planilla.ods.`

(8) Responsabilidades del FS (cont.)

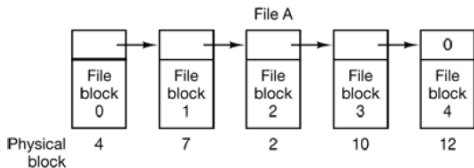
- Más allá de las decisiones que se tomen sobre los puntos anteriores, hay que ver qué pasa tras bambalinas.
- ¿Cómo se representa un archivo? 
- Y dos preguntas relacionadas:
 - ¿Cómo gestiono el espacio libre?
 - ¿Qué hago con los *metadatos*? (ie, los datos sobre los datos: permisos, atributos, etc.).
- Las respuestas a estas preguntas determinan las características del FS, especialmente en cuanto a su rendimiento y confiabilidad.

(9) Representando archivos

- Lo primero que hay que entender es que para el FS un archivo es una lista de bloques + metadata.
- La forma más sencilla de representarlos es poner a los bloques contiguos en el disco.
- Las lecturas son insuperablemente rápidas, pero...
 - ¿Qué pasa si el archivo crece y no tengo más espacio?
 - ¿Qué hago con la fragmentación?
- Nadie usa este esquema en FS de lectoescritura.

(10) El paso obvio

- El paso obvio es usar una lista enlazada de bloques.
- Esto soluciona ambos problemas, pero:
 - Si bien las lecturas consecutivas son razonablemente rápidas, las lecturas aleatorias son muy lentas.
 - Además, desperdicio espacio de cada bloque indicando dónde está el siguiente.



(11) Vuelta de tuerca

- Hay una vuelta de tuerca que se le puede dar a este problema.
- Tengo una tabla que por cada bloque me dice en qué bloque está el siguiente elemento de la lista.
- 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 |

(12) FAT

- FAT usa este método.

| FAT32 | Description |
|---------------------------|----------------------------------|
| 0x?0000000 | Cluster libre |
| 0x?0000001 | Reservado |
| 0x?0000002 - 0x?FFFFFFEF | Clusters de datos |
| 0x?FFFFFFF0 - 0x?FFFFFFF5 | Reservado |
| 0x?FFFFFFF6 | Reservado |
| 0x?FFFFFFF7 | Cluster con sector malo |
| 0x?FFFFFFF8 - 0x?FFFFFFF | Último cluster en archivo (EOC). |

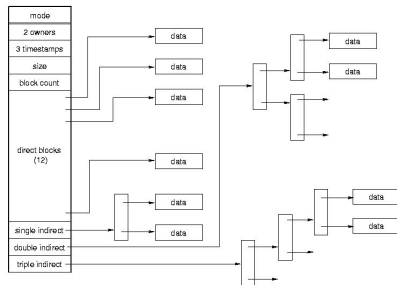
- Soluciona ambos problemas, porque no desperdicio espacio del bloque y porque al tener en memoria todos los bloques que necesito puedo leerlos fuera de orden y no es tan ineficiente para las lecturas no secuenciales.
- Sin embargo, tengo que tener toda la tabla en memoria.
- Puede ser inmanejable para discos grandes.
- Además, única tabla: mucha contención.
- Por otra parte, es poco robusto: si el sistema cae, la tabla estaba en memoria.
- FAT tiene otras limitaciones: no maneja seguridad.

(13) Y... nodos

- La solución Unix son los *inodos*.
- Cada archivo tiene uno.
- En las primeras entradas hay atributos (tamaño, permisos, etc.).
- Luego están las direcciones de algunos bloques, directamente. Esto permite acceder rápidamente a archivos pequeños (si pensamos en bloques de disco de 8 KB, esto permite hasta 96 KB).
- Sigue una entrada que apunta a un bloque llamados *single indirect block*. En este bloque hay punteros a bloques de datos. Eso sirve para archivos de hasta 16 MB.
- A continuación una entrada llamada *double indirect block*, que apunta a una tabla de *single indirect blocks*. Con eso se cubren archivos de hasta 32 GB.
- Le sigue un *triple indirect block*, que apunta a un bloque de *double indirect blocks*. Eso cubre hasta 70 TB.

(14) inodos (cont.)

- Permite tener en memoria sólo las tablas correspondientes a los archivos abiertos.
- Una tabla por archivo → mucha menos contención.
- Consistencia: sólo están en memoria las listas correspondientes a los archivos abiertos.



(15) Implementación de directorios

- ¿Cómo se implementa el árbol de directorios?
- Se reserva un inodo como entrada al *root directory*.
- Por cada archivo o directorio dentro del directorio hay una entrada.
- Dentro del bloque se guarda una lista de (inodos, nombre de archivo/directorio).
- En algunos casos, cuando los directorios son grandes, conviene pensarlos como una tabla de hash más que como una lista lineal de nombres, para facilitar las búsquedas.
- A veces esto se hace a mano, en la capa de software de aplicación.
- ¿Y dónde están los inodos?



(16) inodos - archivos y directorios

File descriptor (inode):

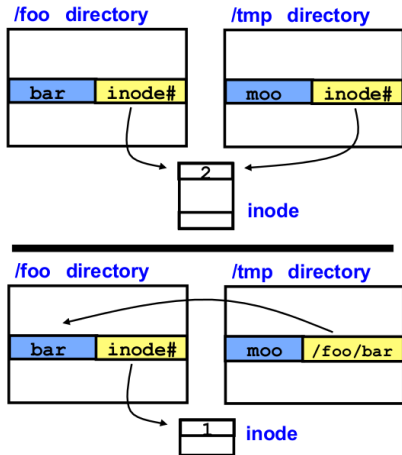
| |
|------------------------------------|
| <code>ulong links;</code> |
| <code>uid_t uid;</code> |
| <code>gid_t gid;</code> |
| <code>ulong size;</code> |
| <code>time_t access_time;</code> |
| <code>time_t modified_time;</code> |
| <code>addr_t blocklist...;</code> |

Directory file:

| | |
|--------------------|--------------|
| Filename | inode# |
| Filename | inode# |
| REALLYLONGFILENAME | |
| inode# | Filename |
| inode# | Short inode# |

(17) inodos - links

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





(18) Atributos

- Cuando se habla de *metadata* en general se incluyen los inodos (o la estructura de datos que sea que use el FS) pero además otra información.
- Por ejemplo:
 - Permisos (default y ACLs).
 - Tamaños.
 - Propietario/s.
 - Fechas de creación, modificación, acceso.
 - Bit de archivado.
 - Tipo de archivo (regular, dispositivo virtual, pipe, etc.).
 - Flags.
 - Conteo de referencias.
 - CRC o similar.

(19) Manejo del espacio libre

- Otro problema es cómo manejar el espacio libre.
- Una técnica posible es utilizar un mapa de bits empaquetado, donde los bits en 1 significan libre.
- Así, si una palabra tiene todos 0 puedo saltarla por completo con una única comparación.
- Pero requiere tener el vector en memoria, y eso no está bueno.
- También podemos tener una lista enlazada de bloques libres.
- En general se clusteriza. Es decir, si un bloque de disco puede contener n punteros a otros bloques, los primeros $n - 1$ indican bloques libres y el último es el puntero al siguiente nodo de la lista.
- Un refinamiento consiste en que cada nodo de la lista indique, además del puntero, cuántos bloques libres consecutivos hay a partir de él.


(20) Caché

- Una manera de mejorar el rendimiento es mediante la introducción de un *caché*. 
- le, una copia en memoria de bloques del disco.
- Se maneja de manera muy similar a las páginas.
- De hecho, los SO modernos manejan un caché unificado para ambas, ya que si no, al mapearse archivos en memoria, tendríamos dos copias de lo mismo.
- Un efecto muy interesante del caché es que puede grabar las páginas de manera ordenada, de manera tal que el administrador de E/S pueda planificar más eficientemente la escritura. 
- A veces, las aplicaciones pueden configurarse para hacer escritura *sincrónica*, es decir, escribiendo en disco inmediatamente.
- Esto es mucho más lento.

(21) Consistencia

- ¿Qué pasa si se corta la energía eléctrica antes de que se graben a disco los cambios?
- Los datos se pierden. Por eso se provee el system call `fsync()`, para indicarle al SO que queremos que las cosas se graben sí o sí. Es decir, que grabe las páginas “sucias” del caché.
- Sin embargo, el sistema podría interrumpirse en cualquier momento.
- La alternativa más tradicional consiste en proveer un programa que restaura la consistencia del FS. En Unix se llama `fsck`.
- Básicamente, recorre todo el disco y por cada bloque cuenta cuántos inodos le apuntan y cuántas veces aparece referenciado en la lista de bloques libres. Dependiendo de los valores de esos contadores se toman acciones correctivas, cuando se puede.

(22) Consistencia (cont.)

- La idea es agregarle al FS un bit que indique apagado normal.
- Si cuando el sistema levanta ese bit no está prendido, algo sucedió y se debe correr fsck.
- El problema es que eso toma mucho tiempo y el sistema no puede operar normalmente hasta que este proceso termine.
- Hay algunas alternativas para evitar eso, total o parcialmente.
- Una se llama *soft updates*: se trata de rastrear las dependencias en los cambios de la metadata para grabar sólo cuando hace falta.
- Sigue haciendo falta una recorrida por la lista de bloques libres, pero se puede hacer mientras el sistema está funcionando.
- Otra: *journaling*. 

(23) Journaling

- Algunos FS, como ReiserFS, ZFS, ext3, NTFS, etc. llevan un *log* o *journal*.
- Le, un registro de los cambios que habría que hacer.
- Eso se graba en un buffer circular. Cuando se baja el caché a disco, se actualiza una marca indicando qué cambios ya se reflejaron.
- Si el buffer se llena, se baja el caché a disco.
- Hay un impacto en performance pero es bajo porque:
 - Este registro se escribe en bloques consecutivos, y una escritura secuencial es mucho más rápida que una aleatoria.
 - Los FS que no hacen journal escriben a disco inmediatamente los cambios en la metadata, para evitar daños mayores en los archivos.
- Cuando el sistema levanta, se aplican los cambios aún no aplicados. Esto es mucho más rápido que recorrer todo el disco.

(24) Características avanzadas

- Otras cosas que puede incluir un FS avanzado:
- Cuotas de disco:
 - Idea: limitar cuánto espacio puede utilizar cada usuario.
 - Notar: puede ser difícil de implementar. No alcanza con poner un contador simple en el `write()` porque puede haber escrituras concurrentes.
- Encriptación:
 - ¿Cómo/dónde guardo la clave?
 - Por ejemplo, CFS y EFS.
- Snapshots:
 - Son “fotos” del disco en determinado momento.
 - Se hacen instáneamente.
 - El SO sólo duplica los archivos que se modifican.
 - Muy bueno para hacer copias de seguridad.

(25) Características avanzadas (cont.)

- Manejo de RAID por software.
 - Desventaja: más lento.
 - Ventaja: mayor control.
 - Independencia de proveedor.
 - A veces, nuevos niveles de redundancia (por ejemplo, ZFS).
- Compresión.

(26) Performance

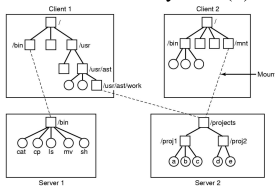
- Muchos factores impactan en el rendimiento:
- Tecnología de disco.
- Política de scheduling de E/S.
- Tamaños de bloque.
- Cachés del SO.
- Cachés de las controladoras.
- Manejo general de locking en el kernel.
- FS
 - Journaling vs. softupdates. La batalla continua...
 - Ver por ejemplo;
http://www.usenix.org/event/usenix2000/general/full_papers/seltzer/seltzer_html/
- ¿Hasta el último nanosegundo de performance? Tal vez pueda sacrificar un poco en pos de mantenibilidad o robustez.

(27) NFS

- El *Network File System* es un protocolo que permite acceder a FS remotos como si fueran locales, utilizando RPC.
- La idea es que un FS remoto se monta en algún punto del sistema local y las aplicaciones acceden a archivos de ahí, sin saber que son remotos.
- Para poder soportar esto, los SO incorporan una capa llamada *Virtual File System*.
- Esta capa tiene *vnodes* por cada archivo abiertos. Se corresponden con inodos, si el archivo es local. Si es remoto, se almacena otra información.
- Así, los pedidos de E/S que llegan al VFS son despachados al FS real, o al *cliente de NFS*, que maneja el protocolo de red necesario.
- Si bien del lado del cliente es necesario un módulo de kernel, del lado del server alcanza con un programa común y corriente.

(28) NFS (cont.)

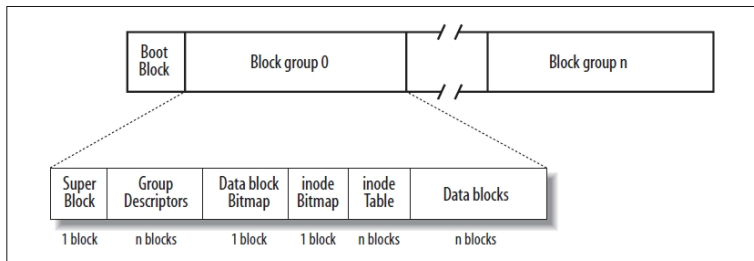
Network File System (1)



- Examples of remote mounted file systems
- Directories are shown as squares, files as circles

- Otros FS distribuidos funcionan de manera similar (en cuanto a su integración con el kernel).
- Notar que, desde cierto punto de vista, NFS no es 100 % distribuido, ya que todos los datos de un mismo “directorio” deben vivir físicamente en el mismo lugar.
- Hay FS 100 % distribuidos, como AFS o Coda.
- Han tenido un éxito relativo.

(29) Estructura de FS Ext2



- El superbloque (superblock) contiene metadatos críticos del sistema de archivos tales como información acerca del tamaño, cantidad de espacio libre y donde se encuentra los datos. Si el superbloque es dañado, y su información se pierde, no podría determinar que partes del sistema de archivos contiene información.

(30) Ext2 - superbloque

| 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 |

(31) Ext2 - superbloque (cont.)

| | | |
|-------------|--------------------------|---|
| __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 |

(32) Ext2 - Group Descriptor

| 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 |

(33) Ext2 - Inode

| 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 |

(34) Algunos números - Ext2

- Supongamos una partición Ext2 de 8GB, con bloques de 4KB.
- El bloque de bitmap de datos (4KB) describe 32K bloques de datos (es decir, 128 MB).
- Entonces, como mucho se requieren 64 grupos de bloques.
- Más info: Libro *Understanding the Linux Kernel - 3rd edition*

- Vimos
 - Responsabilidades del FS.
 - Punto de montaje.
 - Representación de archivos.
 - Manejo del espacio libre.
 - FAT, inodos.
 - Atributos.
 - Directorios.
 - Caché.
 - Consistencia, journaling.
 - Características avanzadas.
 - NFS, VFS.
 - Ext2